Masterarbeit

Untersuchungen zur Modellierung, Algorithmierung und Programmierung von Verkehrsmodellen für die Verkehrssimulation

Lorenz, Mario
geb. am 12.01.1986 in Zwickau

Studiengang Informatik (Master)

Westsächsische Hochschule Zwickau
Fakultät Physikalische Technik/Informatik
Fachgruppe Informatik

Betreuer, Einrichtung:	Prof. Dr. rer. nat. Silke Kolbig, WH Zwickau
	Prof. Dr.-Ing. habil. Wolfgang Kühn, WH Zwickau
Abgabetermin:	28.07.2010

Autorenreferat

Diese Masterarbeit untersucht die Möglichkeiten der virtuellen Simulation von Verkehr, in dem ein vom Menschen gesteuertes Fahrzeug aktiv beteiligt ist.
Es wird dabei vor allem auf mikroskopische Verkehrsmodelle eingegangen, die Grundlage der Simulation sind. Modellierung und Implementierung eines Softwaremoduls, welches als Bestandteil eines Verteilten Systems auf der Basis vorgestellter Verkehrsmodelle den Verkehr simuliert, werden beschrieben. Dabei wird darauf eingegangen, wie die Abarbeitung parallelisiert erfolgen kann.
Untersuchungsergebnisse zu den verschiedenen Verkehrsmodellen werden vorgestellt. Sie geben Aufschluss darüber, wie real Menschen das Fahren in Verkehrsströmen empfinden.
Es ist das Anliegen dieser Arbeit, einen Einblick in die Verkehrssimulation zu geben und mit dem entwickelten Modul einen kleinen Beitrag für eine sichere Straßenplanung zu leisten. Mit dessen Hilfe ist es möglich, geplante Straßen unter realistischen Bedingungen virtuell abzufahren und riskante Stellen im Vorfeld der Bauung zu erkennen.

Danksagung

Ich möchte mich an dieser Stelle bei allen bedanken, die mir bei der Erstellung meiner Masterarbeit geholfen haben.
Insbesondere möchte ich mich bei Frau Prof. Dr. Silke Kolbig und bei Herrn Prof. Dr. Wolfgang Kühn für die Bereitstellung des Themas, die Möglichkeit der Durchführung des Projekts und für ihre Unterstützung bedanken.
Bedanken möchte ich mich weiterhin bei Herrn Prof. Dr. Matthias Richter für seine Hilfe bei der Informationsbeschaffung zu den Verkehrsmodellen.
Ich danke Herrn M. Sc. Ronny Kubik und Herrn Dipl.-Wirtsch.-Inf.(FH) Hendrik Völker für die sehr gute und vertrauensvolle Zusammenarbeit sowie für ihre Anregungen und Hilfe.
Mein Dank gilt auch meinen Probanden Alexander Apel, Christan Kollatsch, Frances Korndörfer, Michael Körner, Matthias Krause und Michael Wahle für Ihre Bereitschaft, bei den Testfahrten mitzuwirken.

Inhaltsverzeichnis

Kurzzeichenverzeichnis

CLI..... Common Language Infrastructure
CM..... CarMaker
DV..... DriverVehicle
Fz..... Fahrzeug
GUI..... Graphical User Interface
ID..... Identifikator
IP..... Internet Protocol
Kfz..... Kraftfahrzeug
KI..... Künstliche Intelligenz
LKW..... Lastkraftwagen
LSA..... Lichtsignalanlage
MATLAB..... MATRIX LABORATORY
ÖPNV..... Öffentlicher Personennahverkehr
PKW..... Personenkraftwagen
SV..... Schwerlastverkehr
TCP..... Transmission Control Protocol
UDP..... User Datagram Protocol
UML..... Unified Modeling Language
USA..... United States of America
VISSIM..... Verkehr in Städten-Simulation
VISUM..... Verkehr in Städten-Umlegung
ZIM..... Zentrales Innovationsprogramm Mittelstand

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einleitung

1.1 Organisatorische Einordnung der vorliegenden Arbeit

Die vorliegende Masterarbeit wurde an der Westsächsischen Hochschule Zwickau im Rahmen einer Kooperation mit der IPG Automotive GmbH[1] im Vorfeld eines ZIM-Projekts mit dem Arbeitstitel „Prototypische Umsetzung eines gesamtheitlichen Verfahrens zum Entwerfen – Prüfen – Abwägen – Bauen von Straßenentwürfen“ erstellt. Durch das Projekt sollen neue Methoden und Produkte für die Straßenplanung und das damit einhergehende Genehmigungsverfahren entwickelt werden. Ein Hauptbestandteil dieses ZIM-Projekts ist die virtuelle Simulation. Geplante Straßen sollen dabei realitätsnah dargestellt und abgefahren werden können. Diese Masterarbeit bezieht sich in diesem Virtual Reality-Projekt auf die Erzeugung von realitätsnahem Verkehr.

1.2 Gegenstand und Motivation

Diese Masterarbeit beschäftigt sich vorwiegend damit, wie bereits existierende Verkehrsmodelle benutzt werden können, um die Simulation von realitätsnahem Verkehr in einem Softwaremodul zu realisieren. Dabei gilt es, ein von einem Menschen gesteuertes Fahrzeug in diesen simulierten Verkehr zu integrieren.
Bisher ist es noch nicht möglich, geplante Straßen vor ihrer Bauung mit einem hohen Realitätsgrad virtuell abzufahren. Unfallschwerpunkte werden so erst ein bis zwei Jahren nach der Fertigstellung der Straße erkennbar. Zu diesem Zeitpunkt sind dann meist keine baulichen Veränderungen mehr möglich. Wenn geplante Straßen aber im Vorfeld ihrer Bauung virtuell abgefahren werden könnten, wäre es möglich, diese Unfallschwerpunkte bereits in der Planungsphase zu erkennen und gegebenenfalls durch Veränderungen des Streckenverlaufs ungefährlicher zu machen. Dies würde zu einer Verringerung von Personen- und Sachschäden führen. Das entstehende Softwaremodul soll dabei den Realitätsgrad der gesamten simulierten Verkehrssituation erhöhen.

1.3 Problematik und Fragestellung

Für die Realisierung dieses Verkehrssimulationsmoduls müssen zunächst verschiedene Verkehrsmodelle untersucht und anschließend informationstechnisch so umgesetzt werden, dass die Verkehrssimulation hinreichend schnell ist. Dabei müssen das extern gesteuerte

[1] Die IPG Automotive GmbH entwickelt die Simulationssoftware für Fahrzeuge CarMaker.

CarMaker-Fahrzeug und die Integration des Moduls in ein Verteiltes System berücksichtigt werden.
Aus dieser Grundproblematik ergeben sich folgende Fragestellungen, die im Verlauf dieser Arbeit beantwortet werden sollen:

1. Welche Verkehrsmodelle gibt es? Wie sind sie zu klassifizieren? Welche eignen sich für diesen Anwendungszweck?
2. Welche Verkehrsmodelle sollen implementiert werden?
3. Welcher Funktionsumfang kann in der vorgegebenen Bearbeitungszeit der Masterarbeit umgesetzt werden?
4. Wie ist das Verkehrssimulationsmodul zu modellieren? Wie kann dabei die möglichst leichte Erweiterbarkeit hinsichtlich zusätzlicher Verkehrsmodelle gewährleistet werden? Wie kann eine möglichst hohe Plattformunabhängigkeit erreicht werden?
5. Wie gliedert sich das Verkehrssimulationsmodul in das bestehende Verteilte Softwaresystem ein? Welche Daten müssen zwischen den verschiedenen Modulen ausgetauscht werden?
6. Wie kann die Berücksichtigung des externen CarMaker-Fahrzeuges bei der Simulation des Verkehrs erfolgen? Wie kann das Modul sinnvoll parallelisiert werden? Welches Zeitverhalten hat es?
7. Welche Parametrisierungsmöglichkeiten können sinnvollerweise angeboten werden, um den flexiblen Einsatz des Verkehrssimulationsmoduls zu gewährleisten?
8. Wie ist die Güte des erzeugten Verkehrs der verschieden Modelle? Wie bewerten Menschen den Realitätsgrad des simulierten Verkehrs, wenn sie in ihm fahren?

1.4 Aufbau der Arbeit

Diese Masterarbeit gliedert sich in sechs Kapitel. Im ersten Kapitel erfolgt die Einordnung der Arbeit. Es werden die Hintergründe und die Motivation für diese Masterarbeit beschrieben und Fragen aufgeworfen, die durch sie beantwortet werden sollen.
Das zweite Kapitel beschäftigt sich mit den theoretischen verkehrstechnischen Grundlagen. Es folgen eine Klassifizierung und Erläuterungen zu den Verkehrsmodellen. Insbesondere werden die mikroskopischen Verkehrsmodelle ausgeführt, da sie in dem zu entwickelnden Verkehrssimulationsmodul zum Einsatz kommen. Weiterhin wird noch ein kurzer Einblick in die Verkehrssimulationstechnik gegeben.
Das dritte Kapitel beschreibt die Anforderungen an das Verkehrssimulationsmodul und seine Modellierung.
Im vierten Kapitel wird ausführlich auf die Implementierung des Moduls eingegangen. Daneben erfolgen eine Beschreibung der Parametrisierungsmöglichkeiten sowie eine Erläuterung zum Zeitverhalten.
Im sich anschließenden fünften Kapitel werden die im Verkehrssimulationsmodul realisierten Verkehrsmodelle verglichen. Nach einer Auswertung der Trajektorien (Bewegungspfade) von

Fahrzeugen im simulierten Verkehr folgt eine Auswertung der Einschätzungen bezüglich des Realitätsgrades durch Testfahrer.
Das abschließende sechste Kapitel beantwortet zunächst die Fragen aus dem ersten Kapitel. Danach werden die Ergebnisse dieser Masterarbeit zusammenfassend beschrieben sowie eine Einschätzung zu den Weiterentwicklungsmöglichkeiten des Verkehrssimulationsmoduls gegeben.

2 Verkehrstechnische Grundlagen

2.1 Überblick

In diesem Kapitel werden die verkehrstechnischen Grundlagen vorgestellt. Zuerst erfolgt die Definition einiger wichtiger Begriffe. Den hauptsächlichen Teil dieses Kapitels nehmen die anschließenden Betrachtungen der Verkehrsmodelltheorie ein. Insbesondere werden dabei die mikroskopischen Verkehrsmodelle erläutert, da diese Grundlage des entwickelten Verkehrssimulationsmoduls sind. Abschließend wird eine kurze Erläuterung zu den wesentlichen Eigenschaften der Verkehrssimulation gegeben.

2.2 Definitionen

2.2.1 Verkehr

Diese Definition bezieht sich auf [Erle07] und [SchLoh97].
Als Verkehr soll die Gesamtheit der Bewegungsvorgänge wie Fahren, Bremsen, Beschleunigen, Überholen, Warten, Anhalten, Einfädeln von Personen und Gütern in Verkehrsmitteln über einem zeitlich und räumlich definiertes Verkehrsgebiet aufgefasst werden. Die Bewegungen der eingesetzten Verkehrsmittel geschehen in der Regel innerhalb klarer Verhaltens- und Bewegungsvorschriften. Verkehr ist meist durch Zweckgebundenheit gekennzeichnet, und seine Bewegung ist auf ein räumliches Ziel hin ausgerichtet. Die Gesamtheit der sich bewegenden Verkehrsmitteleinheiten wird als Verkehrsstrom und die Bewegung selbst als Verkehrsfluss angesehen.

2.2.2 Verkehrsinfrastruktur

Diese Definition bezieht sich auf [Erle07].
Den Verkehrsteilnehmern wird durch eine Verkehrsinfrastruktur ihre Bewegung ermöglicht oder vereinfacht. Der wichtigste Teil der Infrastruktur ist meist ein physisches Fahrwegenetz, durch das eine reibungslose und sichere Fortbewegung der Fahrzeuge ermöglicht wird. Es gibt eine Vielzahl von unterschiedlichen Verkehrsnetzen. Dazu zählen Straßen samt Geh- und Fahrradwege, das Schienennetz, die Kanal- und Flussnetze der Binnenschifffahrt und auch die internationalen Seeschifffahrtsrouten sowie die Luftfahrtkorridore. Als weiterer Teil der Verkehrsinfrastruktur ist die Straßenausstattung mit Lichtsignalanlagen, Beschilderungen, Verkehrsleitsysteme und Markierungen anzusehen. Sie erhöhen die Qualität des Verkehrsablaufes und unterstützen die Verkehrsteilnehmer bei ihrem Bestreben, die Zielorte zu erreichen.

2.2.3 Fahrzeuge

Diese Definition bezieht sich auf [Erle07].
Fahrzeuge vereinfachen und ermöglichen den von Personen und für Güter angestrebten Ortswechsel auf ihren jeweiligen Streckennetzen. Eine analoge Klassifizierung zur Verkehrsinfrastruktur lässt sich auch hier vornehmen. So können die Gruppen der Straßen-, Wasser-, Schienen- und Luftfahrzeuge unterschieden werden.

2.2.4 Dynamische Grundgleichungen eines Fahrzeuges

Die dynamische Grundgleichung beschreibt die Bewegung eines Fahrzeuges, das dabei idealisiert als Punktmasse betrachtet wird. Für ein Fahrzeug, das sich mit einer Beschleunigung von $a(t)$ vom Zeitpunkt t_i bis zum Zeitpunkt t_{i+1} bewegt, gelten für seine Position s und seine Geschwindigkeit v die folgenden Gleichungen:

$$s(t_{i+1}) = s(t_i) + \int_{t_i}^{t_{i+1}} v(t)\, dt \tag{2.1}$$

$$v(t_{i+1}) = v(t_i) + \int_{t_i}^{t_{i+1}} a(t)\, dt \tag{2.2}$$

Falls die Bewegung eines Fahrzeuges in diskreten Zeitschritten betrachtet wird, wie zum Beispiel in einer zeitschritt-basierten Simulation (vergleiche Abschnitt 2.4.3), sind Beschleunigung und Geschwindigkeit für die Dauer eines solchen Zeitschrittes Δt näherungsweise als konstant anzusehen. Die Gleichungen (2.1) und (2.2) gelten dann in folgenden abgewandelten Formen:

$$s(t_{i+1}) = s(t_i) + \Delta t \cdot v(t_{i+1}) \tag{2.3}$$

$$v(t_{i+1}) = v(t_i) + \Delta t \cdot a(t_{i+1}) \tag{2.4}$$

2.2.5 Verkehrsstärke

Diese Definition bezieht sich auf [SchLoh97].
Die Verkehrsstärke M drückt aus, wie viele Fahrzeuge eines Verkehrsstromes innerhalb einer Zeitdauer einen lokalen Messpunkt x_0 passieren. Sie ergibt sich als Quotient aus der Anzahl der während der Messzeit am Messpunkt registrierten Fahrzeuge N und dieser Messzeit Δt:

$$M = \frac{N}{\Delta t} \quad [Fz/h] \tag{2.5}$$

Entscheidend für die Bestimmung der Verkehrsstärke ist die Länge des Messintervalls. Bei sehr kleiner Wahl, zum Beispiel zehn Sekunden, hat ein einzelnes Fahrzeug sehr großen Einfluss auf das Ergebnis, wodurch es zu großen Schwankungen zwischen den einzelnen Messungen kommen kann. Hingegen werden bei großer Messdauer, zum Beispiel zwölf Stunden, Spitzenverkehrsstärken durch die zeitliche Mittelung verloren gehen. Die übliche Länge des Messintervalls liegt zwischen fünf Minuten und einer Stunde.

2.2.6 Verkehrsdichte

Diese Definition bezieht sich auf [SchLoh97].
Die Verkehrsdichte D gibt die Anzahl von Fahrzeugen an, die sich zu einem konkreten Beobachtungszeitpunkt auf einem bestimmten Wegeabschnitt der freien Strecke befinden. Im Gegensatz zur Verkehrsstärke, bei der die Ermittlung über ein Zeitintervall Δt erfolgt. Die Verkehrsdichte ergibt sich als Quotient aus der zum Messzeitpunkt t_0 im Messbereich befindlichen Anzahl an Fahrzeugen N und dem Messbereich Δs:

$$D = \frac{N}{\Delta s} \quad [Fz/km] \tag{2.6}$$

Üblicherweise wird als Messbereich ein Kilometer betrachtet.

2.2.7 Kapazität und Auslastung

Diese Definition bezieht sich auf [SchLoh97].
Als Kapazität oder Durchlassfähigkeit C wird die größtmögliche Verkehrsstärke an einem bestimmten Querschnitt bezeichnet, die ein Verkehrsstrom erreichen kann.

$$C = M_{\max} \quad [Fz/h] \tag{2.7}$$

Diese maximale Verkehrsstärke wird allerdings nur in den Spitzenstunden erreicht. Auf ihrer Basis lässt sich aber ein Auslastungsgrad A definieren, der den gegenwärtigen Anteil der Verkehrsstärke an der Kapazität der Strecke beschreibt.

$$A = \frac{M}{C} \quad [-] \tag{2.8}$$

2.2.8 Mittlere Geschwindigkeit

Diese Definitionen beziehen sich auf [SchLoh97].
Es werden zwei Arten der mittleren Geschwindigkeit unterschieden: die lokale mittlere Geschwindigkeit und die momentane mittlere Geschwindigkeit. Der Unterschied besteht in der Art der Messung. Bei Ermittlung der lokalen mittleren Geschwindigkeit werden an einem festen Streckenquerschnitt die Geschwindigkeiten der Fahrzeuge v_l über eine gewisse Zeitdauer erfasst. Dies kann mit Hilfe eines Radarmessgerätes erfolgen. Wenn N die Anzahl der Messwerte darstellt, gilt für die lokale mittlere Geschwindigkeit $\overline{v_l}$ folgende Gleichung:

$$\overline{v_l} = \frac{1}{N}\sum_{i=1}^{N} v_{li} \tag{2.9}$$

Die mittlere momentane Geschwindigkeit betrachtet die Geschwindigkeiten der Fahrzeuge v_m, die sich zu einem festen Zeitpunkt innerhalb eines bestimmten Streckenabschnitts befinden. Wenn N die Anzahl der Messwerte darstellt, gilt für die momentane mittlere Geschwindigkeit $\overline{v_m}$ folgende Gleichung:

$$\overline{v_m} = \frac{1}{N}\sum_{i=1}^{N} v_{mi} \tag{2.10}$$

2.2.9 Fahrzeugabstand

Diese Definitionen beziehen sich auf [SchLoh97].
Beim Abstand zweier Fahrzeuge wird zwischen dem Bruttoabstand und dem Nettoabstand unterschieden. Der Bruttoabstand dx_{brutto} bezeichnet den Abstand zwischen zwei Vergleichspunkten an den beiden Fahrzeugen, meist ist das die vordere Stoßstange. Der Nettoabstand dx_{netto} hingegen bezeichnet die kürzeste räumliche Distanz zwischen zwei Fahrzeugen, also die Entfernung vom Heck des vorderen Fahrzeuges zum Bug des hinteren Fahrzeuges (siehe Abbildung 2.1).

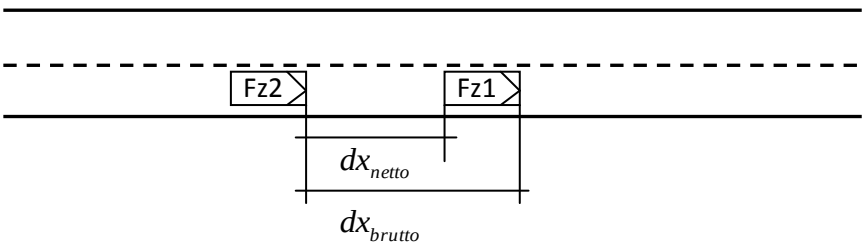

Abbildung 2.1: Brutto- und Nettoabstand aus Sicht von Fz2

2.2.10 Zeitlücke

Diese Definitionen beziehen sich auf [SchLoh97].
Die Zeitlücke bezeichnet den zeitlichen Abstand zweier aufeinanderfolgender Fahrzeuge beim Passieren eines Messquerschnitts. Dabei wird analog zum Fahrzeugabstand zwischen Bruttozeitlücke und Nettozeitlücke unterschieden. Die Bruttozeitlücke dt_{Brutto} ist die Zeit, die zwischen dem Passieren des Bugs des vorausfahrenden Fahrzeuges und dem Bug des hinterherfahrenden Fahrzeuges vergeht. Die Nettozeitlücke dt_{netto} bezeichnet die Zeitspanne zwischen dem Heck des voranfahrenden Fahrzeuges und dem Bug des hinterherfahrenden Fahrzeuges.

2.2.11 Verkehrszustände

Diese Definitionen beziehen sich auf [SchLoh97].
Der Verkehrsablauf kann in vier Zustände unterteilt werden:

1. **Freier Verkehr**: Der Fahrer eines Fahrzeuges kann seine Geschwindigkeit frei wählen und wird dabei nur durch die baulichen Bedingungen und die Limitierungen seines Fahrzeuges beschränkt. Eine Beeinflussung durch andere Fahrzeuge findet nicht statt. Die Verkehrsdichte ist sehr klein.
2. **Teilgebundener Verkehr**: Die Verkehrsdichte hat ein solches Maß erreicht, dass es zu Pulkbildungen kommt. Die Fahrer müssen nun ihre Geschwindigkeit auch von den Geschwindigkeiten der anderen Fahrzeuge abhängig machen.
3. **Gebundener Verkehr**: Es entstehen Fahrzeugkolonnen. Die Fahrer wählen ihre Geschwindigkeiten nun maßgeblich nach denen der anderen Fahrzeuge in der Kolonne. Die Verkehrsdichte ist nun groß.
4. **Überfüllung**: Es kommt zu einem ständigen Abbremsen und Beschleunigen, beziehungsweise Anhalten und Anfahren der Fahrzeuge. Die Verkehrsdichte hat nun ihr Maximum erreicht. Es kommt zum sogenannten Stop-and-Go.

Die Übergänge von einem Zustand in den anderen sind unscharf. Der Verkehrsfluss bei freiem Verkehr bis zum noch kontinuierlich fahrenden Kolonnenverkehr wird als stabil bezeichnet. Wenn es im Kolonnenverkehr zu Stop-and-Go kommt, wird vom instabilen Verkehrsfluss gesprochen.

2.3 Verkehrsmodelle

2.3.1 Einleitung

Bei der Planung und dem Entwurf für Strecken und Streckennetze ist zu berücksichtigen, welche Auswirkungen diese Änderungen im bestehenden Streckennetzsystem hervorrufen. Der Planungs- und Entwurfsingenieur ist immer bestrebt, einen kontinuierlichen Fluss des Verkehrs ohne Stop-and-Go-Wellen oder Staus zu gewährleisten. Das heißt, dass selbst bei hohen Verkehrsstärken eine Verkehrsdichte angestrebt wird, die einen stabilen Verkehrsfluss erlaubt. Deswegen ist es wichtig, Verkehr so realitätsnah wie möglich im Streckennetz zu simulieren. Diese Simulationen basieren auf sogenannten Verkehrsmodellen. So kann etwa mit einer realitätsnahen Simulation überprüft werden, ob eine geplante Umgehungsstraße wirklich die von ihr erwartete Entlastung einer bestehenden Strecke zur Folge hat.

Aber nicht nur bei der Planung und dem Entwurf von Strecken und Streckennetzen ist die Simulation des Verkehrsgeschehens bedeutsam, sondern auch bei der Verkehrssteuerung. Auf Strecken mit Verkehrsleitsystemen kann auf Basis aktueller Verkehrskenngrößen, wie zum Beispiel der Verkehrsdichte, –stärke und der mittleren Geschwindigkeit, das zukünftige Verkehrsaufkommen prognostiziert werden. Durch entsprechende Maßnahmen, wie das Ändern der Richtgeschwindigkeit oder das Freigeben einer zusätzlichen Fahrspur (so vorhanden), lässt sich eventuell ein prognostizierter Stau verhindern. Mittels Verkehrsbeeinflussung ist es möglich, die Unfallhäufigkeit zu senken (vergleiche [Focus10]). Auch an Zufahrten von Strecken wird an manchen Stellen versucht, durch LSA ein Fließen des Verkehrs aufrechtzuerhalten, indem genau dann Fahrzeugen der Zufluss gestattet wird, wenn auf der Strecke Lücken entstehen.[2] Für die intelligente Steuerung dieser LSA werden Erkenntnisse von Verkehrsmodellen herangezogen.

Es existieren viele unterschiedliche Modelle, die Verkehrsfluss nachbilden und simulieren. Die Spannbreite reicht dabei von Modellen, die von physikalischen Gesetzmäßigkeiten beeinflusst sind[3], über verhaltens- und absichtsbasierte Modelle, bis hin zu Fahrzeugfolgemodellen und Zellulare-Automaten-Modellen. Neue Modellansätze[4], Hybridformen und Abwandlungen bekannter Modelle werden auch gegenwärtig weiterhin publiziert. Grundsätzlich lassen sich diese Modelle aber in drei Gruppen einordnen: makroskopische Verkehrsmodelle, mikroskopische Verkehrsmodelle und mesoskopische Verkehrsmodelle. Es soll dabei besonders auf die mikroskopischen Verkehrsmodelle eingegangen werden (siehe Abschnitt 2.3.3), da sie die Basis für das Verkehrssimulationsmodul bilden.

[2] Hierbei handelt es sich hauptsächlich um Autobahnen.

[3] Zu nennen wären hier das fluiddynamische Modell von Lighthill, Whitham und Richards und die auf der Boltzmann'schen kinetischen Gastheorie basierenden mesoskopischen Modelle.

[4] Zum Beispiel in [Helb97] und [TreiHelb02].

2.3.2 Makroskopische Verkehrsmodelle

Um das Verkehrsverhalten in relativ großen Verkehrsnetzen zu simulieren, werden makroskopische Verkehrsmodelle eingesetzt, da ihre Rechenzeit nicht von der Anzahl der Fahrzeuge abhängig ist. Es existieren viele verschiedene makroskopische Modelle. Sie bedienen sich meist Analogien der Dynamik von Flüssigkeiten und basieren nicht auf den einzelnen Fahrzeugen mit ihren Fahreigenschaften, sondern auf der Verkehrsdichte und der Geschwindigkeitsverteilung. Dabei wird die Gesamtheit aller Fahrzeuge auf einem Streckenabschnitt als kontinuierlicher Volumenstrom aufgefasst. Dieser fließt durch das Streckennetz und ist dabei den Gesetzen der Hydrodynamik unterworfen. Das einzelne Fahrzeug wird so zum Atom des Volumenstroms und verliert seine individuellen Eigenschaften. Teilt sich ein Volumenstrom an einem Knotenpunkt auf, geschieht dies anhand der vorherrschenden Druckverhältnisse. Diese Betrachtungsweise hat den Vorteil, dass die Verkehrskenngrößen Verkehrsdichte, Verkehrsstärke und mittlere Geschwindigkeit äquivalent zu den Größen Dichte, Volumenstrom und Strömungsgeschwindigkeit der Strömungsmechanik behandelt werden können.
Als Grundlage der meisten realitätsnahen makroskopischen Modelle dienen Gleichungen für die Verkehrsdichte und die Geschwindigkeitsverteilung, welche auf fluiddynamischen Betrachtungen basieren. Das erste makroskopische Modell ist das fluiddynamische Modell von Lighthill, Whitham und Richards. Weitere anerkannte makroskopische Modelle sind das Paynes-Modell, das Phillips-Modell, das Kühne-Kerner-Kornhäuser-Modell und das Hilliges-Weidlich-Modell. All die genannten Modelle besitzen allerdings eine gemeinsame Struktur (vergleiche [Helb97], Kap. 17.7). Da der Fokus dieser Arbeit auf den mikroskopischen Modellen und deren Einsatz liegt, soll an dieser Stelle nicht weiter auf makroskopische Modelle eingegangen werden.

2.3.3 Mikroskopische Verkehrsmodelle

2.3.3.1 Einleitung

Mikroskopische Verkehrsmodelle betrachten im Gegensatz zu den makroskopischen Modellen jedes einzelne Fahrzeug und den dazugehörigen Fahrer an sich. Eingesetzt werden sie meist für die Simulation von relativ kleinen Streckennetzen und von Knotenpunkten. Speicher- und Rechenzeitbedarf nehmen bei ihnen mit der Anzahl der simulierten Fahrzeuge zu. Fahrer und Fahrzeuge werden jedoch nicht getrennt betrachtet, sondern als Entitäten, welche die atomare Ebene bilden. Im weiteren Verlauf wird deshalb die Bezeichnung „FahrerFahrzeug“ verwendet.
Die mikroskopischen Verkehrsmodelle versuchen, den Verkehrsstrom durch Interaktion von zwei direkt hintereinander fahrenden Fahrzeugen realistisch nachzubilden. Dabei wird davon ausgegangen, dass von allen im Verkehrsstrom befindlichen FahrerFahrzeugen das direkt vorausfahrende den größten Einfluss auf das Fahrverhalten eines Fahrers hat. In einigen

mikroskopischen Verkehrsmodellen werden zum Teil auch noch weiter vorausfahrende Fahrzeuge für die Bestimmung der Reaktion mit herangezogen, beziehungsweise können die Modelle dahingehend adaptiert werden. Natürlich haben daneben auch noch andere Größen Einfluss auf die Reaktion des FahrerFahrzeugs, etwa das Beschleunigungs- und Bremsverhalten des Fahrzeuges.
Makroskopische Betrachtungsgrößen, wie Verkehrsdichte und Verkehrsstärke, können über eine Mittelung der Bewegungsdaten der einzelnen FahrerFahrzeuge ebenfalls gewonnen werden.
Auch unter den mikroskopischen Verkehrsmodellen gibt es eine große Vielfalt an unterschiedlichen Ansätzen. Beispielsweise die auf der Fahrzeugfolgetheorie basierenden Fahrzeugfolgemodelle[5], psycho-physische Abstandsmodelle oder das Zellulare-Automaten-Modell. Seit jüngerer Zeit gibt es auch Ansätze, absichtsbasierte Modelle mit Fuzzy-Reglern und Neuronalen Netzen zu koppeln[6]. Eingehend erläutert werden sollen nur die Fahrzeugfolgemodelle (siehe Abschnitt 2.3.3.2), das Wiedemann-Modell (siehe Abschnitt 2.3.3.3) und eine Erweiterung des Wiedemann-Modells (siehe Abschnitt 2.3.3.4), da diese im Verkehrssimulationsmodul implementiert und untersucht wurden.

2.3.3.2 Fahrzeugfolgemodelle

Die ersten Fahrzeugfolgemodelle wurden in der 1950er Jahren von Reuschel und Pipes entwickelt (beschrieben in [Reus50] und [Pipes53]). Sie gehen davon aus, dass ein Fahrer $n+1$ zum Zeitpunkt t seine Geschwindigkeit $v_{n+1}(t)$ am Ort $x_{n+1}(t)$ so wählt, dass sie ausschließlich vom Abstand $s_{n+1}(t)$ zu seinem Vorderfahrzeug n und dessen Geschwindigkeit $v_n(t)$ zum Zeitpunkt t abhängt.

$$s_{n+1}(t) := x_n(t) - x_{n+1}(t) \tag{2.11}$$

Dieser Abstand soll dabei genau dem geschwindigkeitsabhängigen Sicherheitsabstand $S(v_n(t))$ entsprechen. Wenn S_0 den minimallen Fahrzeugabstand im Stillstand und T_r die Reaktionszeit darstellen, dann gilt:

$$S(v_n(t)) = S_0 + T_r \cdot v_n(t). \tag{2.12}$$

Für den Abstand $s_{n+1}(t)$ der Fahrzeuge $n+1$ und n gilt dann also:

$$s_{n+1}(t) = S(v_n(t)). \tag{2.13}$$

[5] Diese werden auch als Follow-the-Leader Models bezeichnet.
[6] Ein solches Modell wird von Harding in [Hard07] beschrieben.

Dies führt zu der Bewegungsgleichung

$$\frac{dx_{n+1}(t)}{dt} = v_{n+1}(t) = \frac{s_{n+1}(t) - S_0}{T_r}. \tag{2.14}$$

Nach Differenzierung der Bewegungsgleichung (2.14) entsteht die Beschleunigungsgleichung $a_{n+1}(t)$ mit dem Sensitivitätsfaktor $\kappa := \frac{1}{T_r}$:

$$\frac{dv_{n+1}(t)}{dt} = a_{n+1}(t) = \kappa \cdot [v_n(t) - v_{n+1}(t)]. \tag{2.15}$$

Dieses Modell erklärt jedoch nicht die empirisch auftretenden Verkehrsdichtewellen. Schon nach kurzer Zeit stellt sich ein Gleichgewichtszustand ein, bei dem alle FahrerFahrzeuge mit der gleichen Geschwindigkeit fahren. Deswegen wird eine Reaktionszeit τ eingeführt. Das FahrerFahrzeug $n+1$ führt seine Beschleunigung oder Verzögerung nun nicht mehr unmittelbar zum Zeitpunkt t aus, sondern um τ versetzt zum Zeitpunkt $t+\tau$. Die Reaktionszeit τ liegt dabei zwischen 0.5 und 2 Sekunden. Dies führt zu der Abwandlung der Gleichung (2.15):

$$a_{n+1}(t+\tau) = \kappa \cdot [v_n(t) - v_{n+1}(t)]. \tag{2.16}$$

Allgemein lässt sich für die Reaktion eines FahrerFahrzeuges die Gleichung aufstellen:

$$\textit{Reaktion} = \textit{Sensitivität} \cdot \textit{Reiz}. \tag{2.17}$$

Dabei entspricht die *Reaktion* dem Verzögern oder Beschleunigen des FahrerFahrzeuges, die *Sensitivität* dem Sensitivitätsfaktor κ und der *Reiz* der Geschwindigkeitsdifferenz zu seinem Vordermann. Da bei geringem Fahrzeugabstand eine starke Reaktion erfolgen muss und zudem zwischen dem Beschleunigungsvermögen und der Geschwindigkeit ein Zusammenhang besteht, führte dies zu neuen Betrachtungsansätzen für die *Sensitivität* κ. Ein allgemeines Modell, welches die verschiedenen Modellvarianten für die Sensitivitätsbeziehung κ integriert, wurde von Gaziz, Herman und Rothery aufgestellt (veröffentlicht in [GaHeRo61]). Über die Parameter m, l und κ_0 kann die Sensitivitätsbeziehung κ parametrisiert werden.[7]

$$\kappa := \frac{\kappa_0 \cdot [v_{n+1}(t+\tau)]^m}{[x_n(t) - x_{n+1}(t)]^l} \tag{2.18}$$

[7] Die Parameter sind empirisch zu bestimmen.

Daraus ergibt sich nun die Beschleunigungsgleichung:

$$a_{n+1}(t+\tau) = \frac{\kappa_0 \cdot [v_{n+1}(t+\tau)]^m}{[x_n(t) - x_{n+1}(t)]^l} \cdot [v_n(t) - v_{n+1}(t)]. \tag{2.19}$$

Die Fahrzeugfolgemodelle haben jedoch vier Probleme:

1. Sie beschreiben kein Überholen langsamer FahrerFahrzeuge durch schnellere auf mehrspurigen Straßen.
2. FahrerFahrzeuge, bei denen der Abstand zum Vordermann sehr groß ist, werden nicht realistisch beschrieben. Statt sich ihrer Wunschgeschwindigkeit anzunähern, beschleunigen sie gar nicht (vergleiche Gleichung (2.19)).
3. Es kommt zu einem Mitzieheffekt. Das hintere FahrerFahrzeug richtet seine Geschwindigkeit immer an der des vorausfahrenden aus. Dadurch kann es zum Beispiel nie zu einem Abreißen kommen, wenn ein aggressiverer Fahrer vor einem gemächlichen Fahrer fährt (vergleiche Gleichung (2.19)).
4. Die Beschleunigungs- und die Bremsdauer werden als gleich groß angenommen.

2.3.3.3 Wiedemann-Modell

Das Wiedemann-Modell wurde von Wiedemann in seiner Habilitationsschrift 1974 veröffentlicht (siehe [Wied74]). Es handelt sich um ein verhaltensbasiertes psycho-physisches Abstandmodell. Maßgeblich für die Entscheidung, welche Verhaltensweise von einem FahrerFahrzeug ausgeführt werden soll, sind Wahrnehmungsschwellen. Für das Festlegen der Größen dieser Wahrnehmungsschwellen stützt sich Wiedemann auf die Arbeiten von Todosiev, Michaels und Cozan (siehe [MicCoz63], [Mich65] und [Todo63]).

Erläuterung der Grundlagen

Wiedemann stellte fest, dass der Bewegungsablauf eines FahrerFahrzeuges aus einem komplexen Entscheidungsprozess resultiert, in den viele Faktoren einbezogen sind. Es kann aber grundsätzlich zwischen den Situationen des unbeeinflussten Fahrens und des beeinflussten Fahrens unterschieden werden. Ein Fahrer ist dabei immer bestrebt, sich mit seiner individuellen Wunschgeschwindigkeit fortzubewegen, was von den unterschiedlichen Einflüssen in beiden Fahrsituationen abhängt.
Beim unbeeinflussten Fahren spielen für den Entscheidungsprozess des Fahrers die Gegebenheiten der Straße[8], seines Fahrzeuges[9], die Verkehrsordnung[10] sowie Eigenschaften und Motivation des Fahrers[11] selbst eine Rolle. Diese Einflussgrößen auf die Fahrentscheidungen

[8] Zum Beispiel Zustand und geometrischer Verlauf.
[9] Zum Beispiel Beschleunigungsvermögen.
[10] Zum Beispiel eine Geschwindigkeitsbegrenzung.
[11] Zum Beispiel ein aggressiver, übermüdeter Fahrer auf Berufsfahrt.

eines Fahrers sind nicht gleich gewichtet. Dominant sind im Wesentlichen der Charakter und die Motivation des Fahrers sowie die gesetzlichen Einschränkungen auf dem Streckenabschnitt.
Beim beeinflussten Fahren müssen zusätzlich noch die anderen Verkehrsteilnehmer während des Entscheidungsprozesses Berücksichtigung finden. Die Dominanz der Einflüsse durch andere Fahrzeuge auf die Entscheidungen des Fahrers steigt mit zunehmender Verkehrsdichte. Dadurch wird es für ihn unmöglich, seine angestrebte Wunschgeschwindigkeit zu fahren.
Jeder Fahrer hat zudem ein individuelles Sicherheitsbedürfnis, welches ihn veranlasst, einen geschwindigkeitsabhängigen Sicherheitsabstand zu seinem Vorderfahrzeug einzuhalten.
Die Einflussdimensionen werden vom Fahrer nicht objektiv und klar abgegrenzt wahrgenommen und verarbeitet. Sie werden von ihm vielmehr subjektiv und summiert erfasst. Eine weitere wichtige Feststellung von Wiedemann ist, dass die Informationen, die auf den Fahrer einwirken und auf deren Basis er seine Entscheidungen trifft, unvollständig sind und verspätet eintreffen. Damit Informationen aber überhaupt vom Fahrer wahrgenommen werden können, müssen sie eine Reizschwelle überschreiten. Gerade beim Übergang vom unbeeinflussten zum beeinflussten Fahren ist die Fähigkeit eines Fahrers, Bewegungen der anderen Fahrzeuge relativ zur Eigenbewegung der Größenordnung nach richtig einzuschätzen[12], von großer Bedeutung. Fehlschätzungen können sonst zu gefährlichen Situationen oder Unfällen führen, wenn zum Beispiel zu spät zu zögerlich gebremst wird.
Die Grenze für Relativbewegungen, die der Mensch noch wahrnehmen kann, haben Michaels, Cozan und Todosiev in psycho-physischen Untersuchungen versucht zu bestimmen. Sie basierten auf dem Zusammenhang, „dass die psychologische Empfindung eine Funktion des ausgeübten physikalischen Reizes ist“ (aus [Wied74], Seite 13). Damit ein Reiz aber überhaupt wahrgenommen werden kann, muss dessen Veränderung im Vergleich zum ursprünglichen Reiz einen gewissen Schwellwert überschreiten. Ein Fahrer nimmt die Relativbewegung eines anderen Fahrzeuges über die Änderung der Größe des Abbildes dieses Fahrzeuges auf der Retina seiner Augen wahr.[13] Das allerdings nur dann, wenn sich die Wandlung der Größe hinreichend schnell vollzieht.
Wenn sich die Größe des Abbildes des Fahrzeuges auf der Retina ändert, geht das auf eine Veränderung des Winkels, dessen Schenkel die Fahrzeugaußenkanten tangieren, zurück. Hierbei wird angenommen, dass sich die Fahrzeugbreite durch eine Strecke abstrahieren lässt (vergleiche Abbildung 2.2, Seite 15).

[12] Beispielsweise beim Auffahren auf ein viel langsameres Fahrzeug.
[13] Wenn ein Fahrer zweimal in den Rückspiegel schaut, erkennt er über die Größenänderung des Fahrzeuges hinter ihm, ob es sich genähert oder entfernt hat.

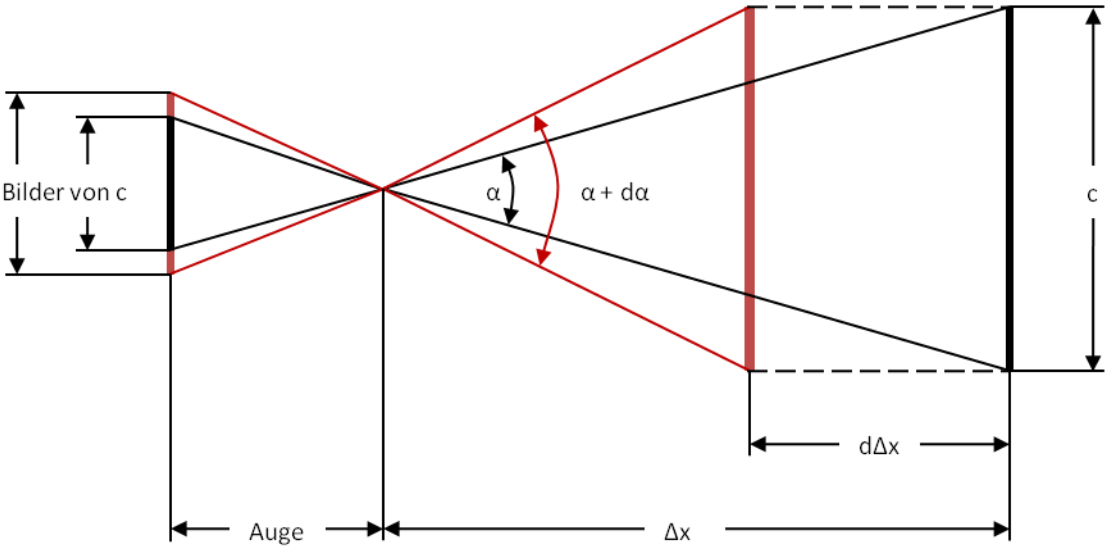

Abbildung 2.2: Wahrnehmung einer Relativbewegungen eines Fahrzeuges durch das Auge (Eigene Darstellung nach [Wied74])

Ausgehend von Abbildung 2.2 werden nun einige Betrachtungen durchgeführt, die schließlich zur Wahrnehmungsschwelle von Relativbewegungen führen.
Für kleine Winkel α gilt:

$$\alpha = \tan(\alpha) = \frac{c}{\Delta x}. \tag{2.20}$$

Nachdem sich c um $d\Delta x$ angenähert hat, beträgt der Winkel α':

$$\alpha' = \alpha + d\alpha = \frac{c}{\Delta x - d\Delta x}. \tag{2.21}$$

Aus den Gleichungen (2.20) und (2.21) lässt sich nun für die Winkeländerung $d\alpha$ folgende Beziehung aufstellen:

$$d\alpha = \frac{c}{\Delta x - d\Delta x} - \frac{c}{\Delta x} = \frac{c \cdot d\Delta x}{\Delta x^2 - \Delta x \cdot d\Delta x}. \tag{2.22}$$

Über die Geschwindigkeitsdifferenz $\Delta v(t)$ der beiden Fahrzeuge kann die Änderung des Abstandes $d\Delta x$ zwischen ihnen während der Zeitspanne dt wie folgt berechnet werden:

$$d\Delta x = \Delta v(t) \cdot dt. \tag{2.23}$$

Wird nun (2.23) in die Gleichung (2.22) eingesetzt, ergibt sich für die Winkeländerung $d\alpha$:

$$d\alpha = \frac{c \cdot \Delta v(t) \cdot dt}{\Delta x^2 - \Delta x \cdot \Delta v(t) \cdot dt}. \tag{2.24}$$

Die Winkeländerungsgeschwindigkeit $\frac{d\alpha}{dt}$ beträgt dann:

$$\frac{d\alpha}{dt} = \frac{c \cdot \Delta v(t)}{\Delta x^2 - \Delta x \cdot \Delta v(t) \cdot dt}. \tag{2.25}$$

Für normale Verkehrssituationen, das heißt kein Unfall, gilt dabei, dass $|\Delta v| << |\Delta d|$ ist. Dadurch kann (2.25) wie folgt vereinfacht werden:

$$\frac{d\alpha}{dt} = \frac{c \cdot \Delta v(t)}{\Delta x^2}. \tag{2.26}$$

Wenn nun die Beziehung (2.26) im Zusammenhang mit den Messergebnissen von Todosiev und Michaels (veröffentlicht in [Mich65] und [Todo63]) für die Wahrnehmungsschwelle von Winkeländerungen zusammengeführt werden, ergibt sich als Wahrnehmungsschwelle für Winkeländerungen, die das Auge erkennen kann:

$$3 \cdot 10^{-4} < \frac{c \cdot \Delta v(t)}{\Delta x^2} < 10 \cdot 10^{-4} \quad [rad/s]. \tag{2.27}$$

Die genaue Größe der Wahrnehmungsschwelle ist allerdings von Mensch zu Mensch unterschiedlich. Sie unterliegt zudem auch bei ein und demselben Menschen (ähnlich der Reaktionszeit) zeitabhängigen Schwankungen.
Das Erkennen von Relativbewegung kann beim Fahrer nicht nur dazu führen, dass er sein Fahrzeug verlangsamt oder beschleunigt, es kann ihn auch zu einem Spurwechsel verleiten.

Modellannahmen

Anhand von Abbildung 2.3, Seite 17 werden nun die Einflüsse der FahrerFahrzeuge aufeinander sowie die verschiedenen Verkehrssituationen beschrieben, wie sie Wiedemann in seinem Modell annimmt. Dabei wird die Möglichkeit des Überholens nicht mit betrachtet.[14]

[14] Wiedemann beschreibt in [Wied74] zwar auch einen theoretischen Ansatz, der Überholvorgänge auf gleichen Richtungsspuren mit einschließt, setzt ihn aber in seinem Algorithmus nicht mit um. Sparmann in [Spar78] und Theis in [Theis97] greifen in ihren Spurwechselmodellen diesen Ansatz jedoch auf. Siehe dazu Abschnitt 2.3.4.

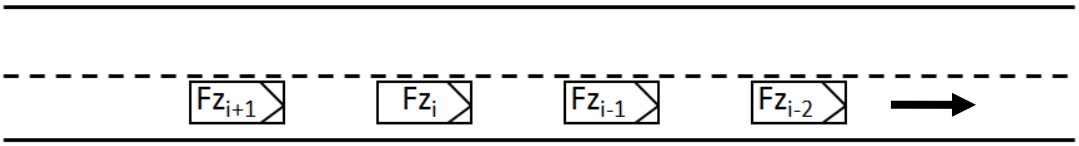

Abbildung 2.3: Indizierungsordnung der Fahrzeuge im Verkehr (Eigene Darstellung nach [Wied74])

Die Betrachtung des Einflusses der FahrerFahrzeuge aufeinander basiert auf Wahrnehmungsschwellen. So wird das FahrerFahrzeug i maßgeblich durch das FahrerFahrzeug $i-1$ beeinflusst. Das FahrerFahrzeug $i-2$ hat bei einem gleichen Geschwindigkeitsniveau ungefähr den doppelten Abstand[15] zu i wie $i-1$. Aus der Beziehung für die Wahrnehmungsschwelle (2.27) folgt nun, dass $i-2$ die vierfache Geschwindigkeitsdifferenz von $i-1$ aufweisen muss, um bei i den gleichen Reiz auszulösen. Somit können alle weiter voraus fahrenden FahrerFahrzeuge vor $i-1$ bei der Beeinflussung des FahrerFahrzeuges i vernachlässigt werden. Auch wenn die Beeinflussung hinsichtlich der Wahrnehmung der Geschwindigkeitsdifferenz nicht mit betrachtet wird, so hat doch das Aufleuchten der Bremslichter des FahrerFahrzeuges $i-2$ innerhalb eines bestimmten Entfernungsbereichs eine nicht vernachlässigbare Einwirkung auf i, die im Modell von Wiedemann mit berücksichtigt ist. Die Beeinflussung von i durch das FahrerFahrzeug $i+1$, etwa durch Drängeln, wird als Sonderfall ausgeschlossen.

Aus diesen Überlegungen heraus werden nun die verschiedenen Verhaltenssituationen aus Sicht des FahrerFahrzeuges i beschrieben.

Wenn sich das FahrerFahrzeug i dem FahrerFahrzeug $i-1$ von hinten nähert, der Abstand aber so groß respektive die Geschwindigkeitsdifferenz so klein ist, dass sie unterhalb der Wahrnehmungsschwelle des Fahrers i liegen, wird er durch $i-1$ in der Wahl seiner Geschwindigkeit nicht beeinflusst. Der Fahrer im Fahrzeug i wird dann seine Wunschgeschwindigkeit anstreben oder, falls er diese bereits erreicht hat, leicht um sie herum pendeln.

Beim Überschreiten der Wahrnehmungsschwelle des Fahrers i beginnt dieser nun mit dem Angleichen seiner Geschwindigkeit an die von $i-1$. Er versucht dabei, die Geschwindigkeit so zu verlangsamen, dass er den von ihm gewünschten Sicherheitsabstand erreicht hat, wenn seine Geschwindigkeit ungefähr der von $i-1$ entspricht. Das Einschätzen, in welchem Maße die Verlangsamung des FahrerFahrzeuges i erfolgt, ist auf Grund ihrer Relativbewegung recht schwierig. Abhängig vom Fahrer ist der auftretende Schätzfehler unterschiedlich groß. Je länger das Abbremsen auf den gewünschten Sicherheitsabstand dauert, desto besser ist er aber in der Lage, die Relativbewegung einzuschätzen und die Verzögerungsrate anzupassen.

Sobald der Fahrer des Fahrzeuges i seine Geschwindigkeit an die des FahrerFahrzeuges $i-1$ angepasst und er den von ihm gewünschten Sicherheitsabstand erreicht hat, befindet er sich im Folgevorgang. Seine Wahrnehmungsschwelle ist nun wieder unterschritten. Wiedemann benutzt den Begriff der „toten Zone“ bezüglich der Wahrnehmung von Relativbewegungen des Fahrers i in dieser Folgesituation. Beim Folgen pendelt das FahrerFahrzeug i leicht um seinen

[15] Im Folgenden ist mit „Abstand“, sofern nicht anders erwähnt, immer der Bruttoabstand gemeint.

Sicherheitsabstand hinter dem FahrerFahrzeug $i-1$. Dies ist dadurch zu erklären, dass der Fahrer i nur noch mit der geringst möglichen steuerbaren Gaspedalbewegung beschleunigt oder verzögert. Durch die Pendelbewegungen von i und von $i-1$ kann es dazu kommen, dass sich das FahrerFahrzeug i zu stark annähert und so das Sicherheitsbedürfnis seines Fahrers unterschritten wird. Er vergrößert dann seinen Abstand wieder durch bewusstes Verzögern. Genauso kann es auch zu einem Abdriften des FahrerFahrzeuges i kommen. Der Fahrer merkt das aber erst, wenn der Abstand seinen eigentlich gewünschten Sicherheitsabstand schon weit überschritten hat. Durch bewusstes Beschleunigen verringert er den Abstand daraufhin wieder.
Dieses Abdriften im Folgevorgang ist nach längerer Dauer oft zu beobachten. Begründet ist dies dadurch, dass die Wahrnehmungsschwellen für das Erkennen von positiven und negativen Geschwindigkeitsdifferenzen unterschiedlich sind. Der Grund dafür lässt sich leicht aus (2.26) beziehungsweise (2.27) erkennen. Der Betrag der Geschwindigkeitsdifferenz muss bei zunehmendem Abstand größer sein als bei abnehmendem, um wahrgenommen zu werden. Ist der Abstand der FahrerFahrzeuge größer als 150 Meter, bewirkt das Überschreiten der Wahrnehmungsschwelle jedoch keine Reaktion des Fahrers.

FahrerFahrzeug-Modell und Algorithmus

Wiedemann nimmt an, dass Schätzvermögen und Verhaltensweisen von Fahrern auf natürliche Weise verteilt sind. Deswegen werden für die Verteilung der nachfolgend erläuterten Größen Normalverteilungen angenommen. Die Größen sind also um den Erwartungswert μ mit der Standardabweichung σ normalverteilt. Ein FahrerFahrzeug hat die folgenden zeitlich unveränderlichen Kenngrößen:

- **Wunschgeschwindigkeit** v_w, (μ,σ)-normalverteilt. Sie stellt die Geschwindigkeit dar, welche ein Fahrer ständig bestrebt ist zu erreichen. Die Wunschgeschwindigkeit hängt im Wesentlichen von der Streckenbeschaffenheit, dem Charakter des Fahrers, den Fahrzeugcharakteristiken und den Rechtsvorschriften ab.
- **Sicherheitsbedürfnis** z_{sb}, $(0.5,0.15)$-normalverteilt.[16] Es beschreibt das Sicherheitsbedürfnis eines Fahrers. Je höher dieses bei einem Fahrer ausgeprägt ist, desto größer wird sein Sicherheitsabstand zu einem vorausfahrenden FahrerFahrzeug sein.
- **Schätzvermögen** z_{sv}, $(0.5,0.15)$-normalverteilt. Es charakterisiert, wie gut ein Fahrer in der Lage ist, Abstände und Geschwindigkeitsdifferenzen einzuschätzen. Diese Größe widerspiegelt die unterschiedlichen Wahrnehmungsschwellen von Fahrern.
- **Beschleunigungswille** z_{bw}, $(0.5,0.15)$-normalverteilt. Er beschreibt das unterschiedliche Beschleunigungs- und Verzögerungsvermögen der Fahrzeuge und den Willen des

[16] Die Angaben der Normalverteilungen in dieser Masterarbeit erfolgen nach dem Schema (Erwartungswert, Standardabweichung).
Alle Werte für Normalverteilungen in diesem Abschnitt stammen aus [Wied74] und haben sich bei den Modellberechnungen als sinnvoll erwiesen.

Fahrers, diese auszunutzen. Dabei wird impliziert, dass ein hoher Beschleunigungswille des Fahrers mit einer hohen maximalen Beschleunigung korrespondiert. Ein aggressiver Fahrer zeichnet sich durch einen hohen Beschleunigungswillen aus.

- **Gaspedalkontrolle** z_{gk}, $(0.5,0.15)$-normalverteilt. Sie kennzeichnet die minimale Beschleunigung, die mit dem Gaspedal des Fahrzeuges kontrolliert werden kann. Ist diese hoch, beschleunigt das Fahrzeug unter Umständen stärker, als es der Fahrer beabsichtigt hat.
- **Bremsvermögen** z_{bv}, $(0.5,0.15)$-normalverteilt. Es stellt dar, ab welcher Verzögerung die Bremslichter eines Fahrzeuges aufleuchten. Wenn ein Fahrer die Bremslichter des Übernächsten sieht, ist er bereits vorgewarnt und kann dann präziser auf die folgende Geschwindigkeitsanpassung seines direkten Vordermanns reagieren.

Außer der Wunschgeschwindigkeit stellen die eben beschriebenen Größen noch nicht die eigentlichen Schwellwerte dar. Sie werden nur als Zufallsfaktor bei ihrer Bestimmung verwendet.

Zu Beginn eines jeden Simulationsschrittes werden zunächst die Wahrnehmungsschwellen des betrachteten FahrerFahrzeuges i berechnet. Ihre Bestimmung erfolgt im Wesentlichen durch den Abstand DX und die Geschwindigkeitsdifferenz DV zu seinem Vordermann $i-1$. Ausgehend von Abbildung 2.4 sind im Folgenden die Schwellen erläutert, welche die Verhaltenszustände des FahrerFahrzeuges abgrenzen: das unbeeinflusste Verhalten (weiß), das bewusst beeinflusste Verhalten (hellgrau) und das unbewusst beeinflusste Verhalten[17] (dunkelgrau).

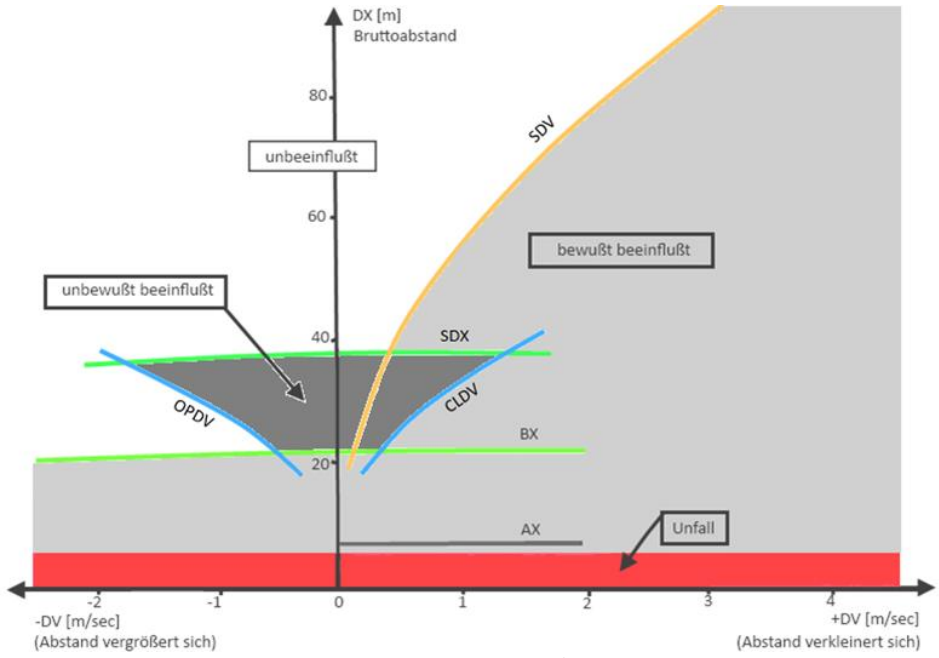

Abbildung 2.4: Beispiel für Wahrnehmungsschwellen[18] (Eigene Darstellung nach [Wied74])

[17] Dies ist die vormals bezeichnete „tote Zone".

[18] Das Diagramm gilt hinsichtlich eines Fahrers mit mittleren Eigenschaften und für ein Geschwindigkeitsniveau von 20 m/s des vorausfahrenden Fahrzeuges.

- **AX.** Bezeichnet den minimalen, vom Fahrer i gewünschten Bruttoabstand zum FahrerFahrzeug $i-1$ im Stillstand. Der Nettoabstand zwischen den FahrerFahrzeugen soll 1 Meter bis 3 Meter betragen. Da dieser Abstand auf das Sicherheitsbedürfnis eines Fahrers zurückgeht, wird für die Berechnung von AX die normalverteilte Zufallsgröße $z_{sb}(i)$ des FahrerFahrzeugs i verwendet. Ferner wird von einer Fahrzeuglänge von 4.5 Metern ausgegangen. AX berechnet sich dann wie folgt:

$$AX = 4.5+1+2\cdot z_{sb}(i)\,. \tag{2.28}$$

- **BX.** Bezeichnet den minimalen vom Fahrer i gewünschten Bruttoabstand zum FahrerFahrzeug $i-1$ bei etwa gleicher Geschwindigkeit der FahrerFahrzeuge während der Fahrt. Es ist vergleichbar mit dem Sicherheitsabstand, den der Fahrer i halten möchte. Bekanntermaßen ist der gefahrene Sicherheitsabstand nicht proportional zur Geschwindigkeit, wodurch bei höheren Geschwindigkeiten relativ kleinere, riskantere Abstände eingehalten werden als bei niedrigeren. Da auch der Abstand BX auf dem Sicherheitsbedürfnis eines Fahrers beruht, wird für seine Berechnung wieder die normalverteilte Zufallsgröße $z_{sb}(i)$ des FahrerFahrzeugs i verwendet. In die Berechnung gehen ferner der Abstand AX und die gefahrene Geschwindigkeit v ein. BX liegt dabei zwischen $AX+\sqrt{v}$ und $AX+8\cdot\sqrt{v}$. Wenn der Abstand abnimmt, wird die Geschwindigkeit $v(i-1)$ verwendet und bei zunehmendem Abstand die Geschwindigkeit $v(i)$. Für die Berechnung von BX gilt:

$$BX = AX+\left(1+7\cdot z_{sb}(i)\right)\cdot\sqrt{v}\,. \tag{2.29}$$

- **SDV.** Bezeichnet bei relativ großen Abständen die Wahrnehmungsschwelle für Geschwindigkeitsdifferenzen. Sie bestimmt, ab wann ein Fahrer anfängt, auf das vorausfahrende Fahrzeug zu reagieren. SDV soll zu gleichen Teilen von der normalverteilten Zufallsgröße $z_{sb}(i)$ und der normalverteilten Zufallsgröße $z_{sv}(i)$ des FahrerFahrzeuges i abhängen. Der Abstand DX, ab dem ein Fahrer reagiert, hängt von seinem Schätzvermögen für die Geschwindigkeitsdifferenz und von seinem Sicherheitsbedürfnis ab. Sind beide groß, reagiert er schon ab einem relativ weiten Abstand. Die Wahrnehmungsschwelle für die Geschwindigkeitsdifferenz DV liegt zwischen $25\cdot\sqrt{DV}$ und $75\cdot\sqrt{DV}$. SDV wird wie folgt berechnet:

$$SDV = \left(\frac{DX-AX}{25\cdot\left(1+z_{sb}(i)+z_{sv}(i)\right)}\right)^2\,. \tag{2.30}$$

- **SDX.** Bezeichnet während des Folgevorganges die obere Grenze, bei deren Überschreiten es zu einem Abdriften des betrachteten FahrerFahrzeuges i kommt. Diese Grenze ist nicht nur zwischen den Fahrern verschieden, sie schwankt mit der Zeit auch bei ein und demselben Fahrer. Deswegen wird für die Berechnung von SDX neben der normalverteilten Zufallsgröße $z_{sv}(i)$ zusätzlich die $(0.5,0.15)$-normalverteilte Zufallsgröße z_n verwendet. SDX unterliegt dabei einem Schwankungsbereich vom 1.5-fachen bis 2.5-fachen um den Mindestabstand der FahrerFahrzeuge i und $i-1$. Hat ein Fahrer also ein gutes Schätzvermögen, ist bei ihm ein Abdriften in geringerem Maße zu beobachten. SDX wird wie folgt berechnet:

$$SDX = AX + (2 - z_{sv}(i) + z_n) \cdot (BX - AX) . \tag{2.31}$$

- **CLDV.** Bezeichnet bei kleinen, abnehmenden Abständen die Wahrnehmungsschwelle des Fahrers i für minimalste Geschwindigkeitsdifferenzen. Wird sie überschritten, löst das eine Reaktion seinerseits aus. Bei CLDV kommen die gleichen Einflüsse wie bei SDX zum Tragen, weswegen für die Berechnung wieder die normalverteilte Zufallsgröße $z_{sv}(i)$ des FahrerFahrzeuges i und die $(0.5,0.15)$-normalverteilte Zufallsgröße z_n herangezogen werden. CLDV lässt sich aus SDV bestimmen, wobei ihr Schwankungsbereich relativ groß ist. Berechnet wird CLDV folgendermaßen:

$$CLDV = SDV \cdot (2 - z_{sv}(i) + z_n)^2 . \tag{2.32}$$

- **OPDV.** Bezeichnet bei kleinen, zunehmenden Abständen die Wahrnehmungsschwelle des Fahrers i für minimalste Geschwindigkeitsdifferenzen. Ist sie überschritten, löst das eine Reaktion seinerseits aus. OPDV berechnet sich aus CLDV, wobei dessen Schwankungsbereich noch größer ist[19] und zwischen dem 1-fachen bis 3-fachen von CLDV liegt. Bei der Berechnung von OPDV wird die $(0.5,0.15)$-normalverteilte Zufallsgröße z_n verwendet. OPDV berechnet sich wie folgt:

$$OPDV = CLDV \cdot (-1 - 2 \cdot z_n) . \tag{2.33}$$

Anhand der eben beschriebenen Schwellen, dem Abstand zwischen den FahrerFahrzeugen i und $i-1$ sowie deren Geschwindigkeitsdifferenz wird gemäß des Entscheidungsbaumes in Abbildung 2.5, Seite 22 während jedes Simulationsschritts bestimmt, welche Reaktion das FahrerFahrzeug i ausführt.

[19] Dies geht aus den vorangestellten Betrachtungen für die Wahrnehmungsschwelle des Auges, also der Beziehung (2.27), hervor.

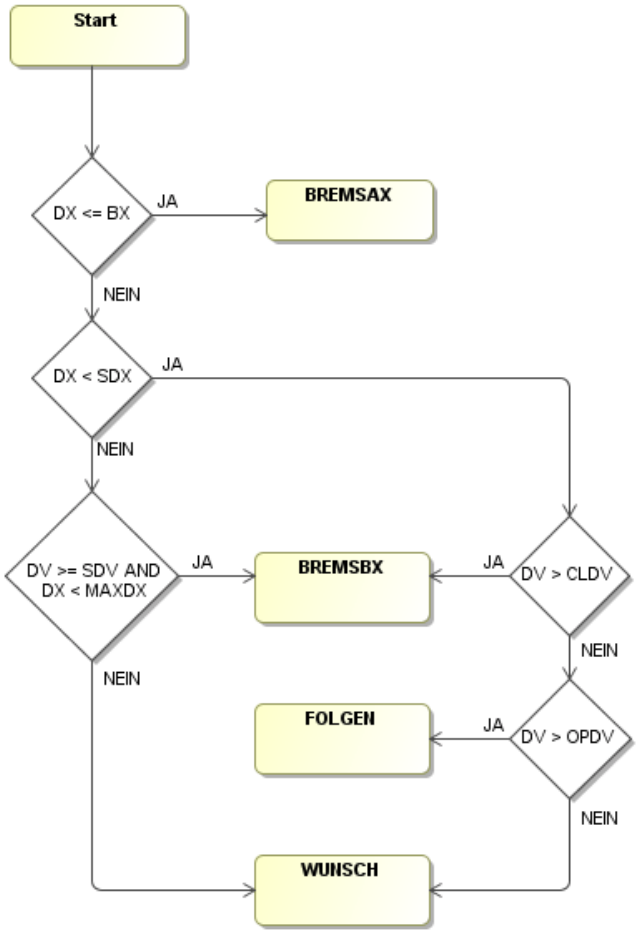

Abbildung 2.5: Entscheidungsbaum für die FahrerFahrzeugreaktion[20] (Eigene Darstellung nach [Wied74])

Die vier Reaktionen korrespondieren mit den drei Zuständen, in denen sich ein FahrerFahrzeug befinden kann. WUNSCH entspricht dem unbeeinflussten Fahren, BREMSBX und BREMSAX dem bewusst beeinflussten Fahren und FOLGEN dem unbewusst beeinflussten Fahren. Ein Vergleich der Abbildung 2.4, Seite 19 mit der Abbildung 2.5 lässt diese Korrespondenz erkennen. Bevor jedoch die Auswirkungen der vier Reaktionen beschrieben werden, müssen zunächst noch einige Größen definiert werden, die hierbei eine Rolle spielen.

Die maximale Beschleunigung $B_{\max}(i)$ des FahrerFahrzeuges i hängt von der momentanen Geschwindigkeit des Fahrzeuges und dem Beschleunigungswillen seines Fahrers ab. Für die Berechnung dieser Größe werden die Geschwindigkeit $v(i)$ und die normalverteilte Zufallsgröße $z_{bw}(i)$ des FahrerFahrzeuges i verwendet. $B_{\max}(i)$ berechnet sich wie folgt:

$$B_{\max}(i) = (0.2 + 0.8 \cdot z_{bw}(i)) \cdot (7 - \sqrt{v(i)}) \quad [m/s^2]. \tag{2.34}$$

[20] MAXDX entspricht mit 150 Metern der Schwelle, ab der das Erkennen von Relativbewegungen kein Reagieren des Fahrers auf das Überschreiten seiner Wahrnehmungsschwelle auslöst.

Für die minimale Beschleunigung $B_{\min}(i)$ des FahrerFahrzeuges i gilt ein analoger Zusammenhang wie für $B_{\max}(i)$. $B_{\min}(i)$ berechnet sich wie folgt:

$$B_{\min}(i) = -8 - 2 \cdot z_{bw}(i) + 0.5 \cdot \sqrt{v(i)} \quad [m/s^2]. \tag{2.35}$$

Die untere Schwelle für die noch steuerbare Beschleunigung und Verzögerung $B_{null}(i)$ des FahrerFahrzeuges i ist abhängig von dessen Gaspedalkontrolle. Sie schwankt in den Grenzen von 0 m/s² bis 0.4 m/s² um 0.2 m/s². Für die Berechnung von $B_{null}(i)$ wird neben der normalverteilten Zufallsgröße $z_{gk}(i)$ des FahrerFahrzeuges i die $(0.5,0.15)$-normalverteilte Zufallsgröße z_n mit herangezogen. Für $B_{null}(i)$ gilt dann:

$$\pm B_{null}(i) = 0.2 \cdot (z_{gk}(i) + z_n) \quad [m/s^2]. \tag{2.36}$$

Für die Reaktion eines Fahrers bedeutet es einen Unterschied, ob das vor ihm fahrende Fahrzeug nur mit der Motorbremse verzögert oder ob dies mit der Bremse geschieht und so dessen Bremslichter aufleuchten. Die Grenze für das Aufleuchten der Bremslichter $B_r(i)$ eines FahrerFahrzeuges i schwankt zwischen -0.5 m/s² und -2.0 m/s². Für die Berechnung dieses Schwellwerts wird die normalverteilte Zufallsgröße $z_{bv}(i)$ des FahrerFahrzeuges i verwendet. $B_r(i)$ berechnet sich wie folgt:[21]

$$B_r(i) = -0.5 - 1.5 \cdot z_{bv}(i) \quad [m/s^2]. \tag{2.37}$$

Die Größe $R(i)$ drückt das besserwerdende Schätzvermögen eines Fahrers i während des Verzögerungsvorgangs aus. Wo sie eingesetzt und wie ihr Wert festgelegt wird, erfolgt bei der Beschreibung der Reaktionen eines FahrerFahrzeuges.
Welche Auswirkungen die vier Reaktionen im Einzelnen haben, soll nun erläutert werden.

- **WUNSCH.** Die Größe $R(i)$ wird zunächst auf 0 gesetzt. Der betrachtete Fahrer i strebt seine individuelle Wunschgeschwindigkeit an oder fährt mit dieser. Dabei ist es ihm jedoch unmöglich, sie exakt konstant zu fahren. Nach Erreichen der Wunschgeschwindigkeit pendelt er mit $+B_{null}(i)$ beziehungsweise $-B_{null}(i)$ um diese. Wenn das FahrerFahrzeug i durch Abdriften aus dem Folgevorgang in WUNSCH geraten ist, beschleunigt es realistischerweise nicht direkt mit dem maximalen Beschleunigungsvermögen $B_{\max}(i)$. Solange der Abstand des FahrerFahrzeuges i zu seinem Vordermann $i-1$ noch kleiner als $2 \cdot BX$ ist, wird die Beschleunigung $B(i)$ wie folgt gesetzt:

[21] Dabei wird der benutzte Gang jedoch nicht mit betrachtet.

$$B(i) = B_{\max}(i) \cdot \frac{DX - BX}{BX} \quad [m/s^2]. \tag{2.38}$$

Dabei gilt, dass für $DX = BX$ das FahrerFahrzeug mit $B(i) = B_{null}(i)$ und bei $DX = 2 \cdot BX$ mit $B(i) = B_{\max}(i)$ beschleunigt. Für den Fall, dass der Fahrer i im letzten Simulationsschritt mit dem Bremspedal gebremst hat, wird gar nicht beschleunigt.

- **BREMSBX.** Die Größe $R(i)$ wird zunächst um 1 erhöht. Der betrachtete Fahrer i erkennt die Annäherung an das FahrerFahrzeug $i-1$ und beginnt mit der Verzögerung auf seinen gewünschten Sicherheitsabstand. Die allgemeine kinematische Gleichung für diesen Vorgang ist:

$$B(i) = \frac{0.5 \cdot DV^2}{BX - DX} + B(i-1) \quad [m/s^2]. \tag{2.39}$$

Dabei wird die Verzögerung $B(i-1)$ des FahrerFahrzeuges $i-1$ nur dann für die Berechnung der Verzögerung $B(i)$ mit berücksichtigt, wenn der Abstand der FahrerFahrzeuge $DX < 2 \cdot BX$ ist und der Fahrer $i-1$ das Bremspedal betätigt. Der Grund dafür ist, dass i die Verzögerung von $i-1$ nur bei relativ kleinem Abstand schätzen kann und er auf das Aufleuchten der Bremslichter stärker reagiert als bei einer reinen Motorbremsung. Der Fahrer i muss die Verzögerung $B(i)$ allerdings schätzen. Dabei verschätzt sich der Fahrer maximal um 1 m/s² in positive beziehungsweise negative Richtung um die objektiv notwendige Verzögerung $B(i)$. Auf die Größe des Schätzfehlers hat das Schätzvermögen des Fahrers Einfluss. Zudem wird sein Schätzfehler kleiner, je länger er sich in der Verzögerung befindet. Für die Berechnung des Schätzfehlers $\Delta B(i)$ werden die normalverteilte Zufallsgröße $z_{sv}(i)$ des FahrerFahrzeuges i, die $(0.5, 0.15)$-normalverteilte Zufallsgröße z_n und die Größe $R(i)$ verwendet. $\Delta B(i)$ berechnet sich wie folgt:

$$\Delta B(i) = \frac{(1 - z_{sv}(i)) \cdot (1 - 2 \cdot z_n)}{R(i)} \quad [m/s^2].^{22} \tag{2.40}$$

Bei längerer Verzögerung wird der Schätzfehler durch wachsendes $R(i)$ immer kleiner. Für die Verzögerung $B(i)$ FahrerFahrzeuges i gilt dann:

[22] Da immer gilt $R(i) \geq 0$ und $R(i)$ zu Beginn von BREMSBX um 1 erhöht wird, gibt es für (2.40) immer eine gültige Lösung.

$$B(i) = \left(\frac{0.5 \cdot DV^2}{BX - DX} + B(i-1) \right) + \Delta B(i) \quad [m/s^2]. \text{[23]} \tag{2.41}$$

Ist die so ermittelte Verzögerung des FahrerFahrzeuges i kleiner als die maximal mögliche Verzögerung $B_{\min}(i)$, wird $B(i) = B_{\min}(i)$ gesetzt. Wenn die Verzögerung kleiner als $-B_{null}(i)$ sein sollte, wird $B(i) = -B_{null}(i)$ gesetzt.

- **FOLGEN.** Der betrachtete Fahrer i befindet sich im Folgevorgang hinter dem FahrerFahrzeug $i-1$. Er ist jetzt in der „toten Zone“ und nimmt keine Relativbewegungen mehr wahr. Das FahrerFahrzeug i beschleunigt ausschließlich um $\pm B_{null}(i)$. Wenn im letzten Simulationsschritt eine positive Beschleunigung vorgelegen hat, beschleunigt das FahrerFahrzeug i jetzt mit $B(i) = B_{null}(i)$, bei negativer Beschleunigung im letzten Simulationsschritt mit $B(i) = -B_{null}(i)$. Der Fahrer i ist zwar in dieser Situation nicht in der Lage Relativbewegungen wahrzunehmen, er kann jedoch andere Signale, wie aufleuchtende Bremslichter, erkennen. Der Fahrer i stellt sich darauf ein, bald selbst bremsen zu müssen, wenn die Bremslichter des noch weiter vorrausfahrenden FahrerFahrzeuges $i-2$ aufleuchten. Das gilt jedoch nur, wenn der Abstand DX' zwischen dem FahrerFahrzeug $i-1$ und dem FahrerFahrzeug $i-2$ kleiner dem BX des betrachteten FahrerFahrzeuges i ist. In diesem Fall wird die Größe $R(i)$ um 1 inkrementiert, anderenfalls $R(i) = 0$ gesetzt.
- **BREMSAX.** Die Größe $R(i)$ wird zunächst um 1 erhöht. Der betrachtete Fahrer i hat sich so stark dem vorausfahrenden FahrerFahrzeug $i-1$ genähert, dass sein Sicherheitsbedürfnis verletzt ist. Seine Reaktion ist dabei umso stärker, je deutlicher sein Sicherheitsabstand unterschritten wurde. Maßgeblich für die Reaktion des Fahrers i in dieser Situation ist, dass er einen Unfall vermeiden und seinen gewünschten Sicherheitsabstand wieder erreichen will. Der Zielabstand, auf den er verzögert, ist AX. Damit kann er gerade noch einen Unfall vermeiden. Hinzukommt, dass das FahrerFahrzeug i nicht unbedingt mit der maximal möglichen Verzögerung $B_{\min}(i)$ bremsen muss. Aus diesen Überlegungen heraus ergibt sich für die notwendige Verzögerung $B(i)$:

$$B(i) = \left(\frac{0.5 \cdot DV^2}{AX - DX} + B(i-1) \right) + B_{\min}(i) \cdot \frac{BX - DX}{BX - AX} \quad [m/s^2]. \tag{2.42}$$

[23] In [Wied74], Seite 45 wird die Gleichung (2.40) nicht mit $+\Delta B(i)$ sondern mit $\cdot \Delta B(i)$ zu Gleichung (2.41) erweitert. Dies ist offenkundig falsch. Im ALGOL–Programmcode [Wied74], Seite 81, Codezeile 739 steht dann auch richtigerweise $+\Delta B(i)$.

Der eingeklammerte Teil von (2.42) entspricht wieder der kinematischen Verzögerungsgleichung auf ein bewegtes Ziel, welche schon in (2.39) verwendet wurde. Der andere Teil der Gleichung (2.42) bezieht sich darauf, dass das FahrerFahrzeug i nicht mit der vollen Verzögerung $B_{\min}(i)$ bremsen muss. Für $DX = BX$ wird dieser Teil 0, bei $DX = AX$ geht $B_{\min}(i)$ voll ein. Aus den gleichen Gründen, die im Punkt BREMSBX aufgeführt wurden, unterliegt das aus (2.42) ermittelte Ergebnis einem Schätzfehler des Fahrers i. Wodurch er auch hier nicht direkt um das $B(i)$ aus Gleichung (2.42) verzögern wird, sondern noch zusätzlich um den Schätzfehler $\Delta B(i)$ aus Gleichung (2.40). Ist die ermittelte Verzögerung des FahrerFahrzeuges i kleiner als die maximal mögliche Verzögerung $B_{\min}(i)$, wird um $B_{\min}(i)$ verzögert. Wenn sie größer als $-B_{null}(i)$ sein sollte, wird die Verzögerung auf $-B_{null}(i)$ gesetzt.

Ein Unfall drückt sich dadurch aus, dass der Abstand DX kleiner als die Fahrzeuglänge von $i-1$ geworden ist. Die Reaktion darauf kann unterschiedlich sein. Zum Beispiel könnte die Wunschgeschwindigkeiten beider FahrerFahrzeuge auf 0 gesetzt werden und so ein Stau entstehen. Die Fahrzeuge können aber auch aus der Simulation entfernt werden oder einfach weiterfahren. Welche Alternative zu bevorzugen ist, hängt im Wesentlichen vom Einsatzzweck und vom Untersuchungsgegenstand der Simulation ab.

Während der Erläuterung der Reaktionsoptionen eines FahrerFahrzeuges wurden zwei Probleme noch nicht beachtet, auf die jetzt eingegangen werden soll: das Schätzen des Abstandes DX, der Geschwindigkeitsdifferenz DV,der Beschleunigung $B(i-1)$ durch den Fahrer i sowie die Reaktionszeit des Fahrers i.

Für die Bestimmung der Beschleunigung $B(i)$ des FahrerFahrzeuges i werden in den meisten Fällen der Abstand DX beziehungsweise die Geschwindigkeitsdifferenz DV der FahrerFahrzeuge i und $i-1$ sowie die Beschleunigung $B(i-1)$ verwendet. Dies sind jedoch objektive Größen. Würden sie benutzt, ergäbe sich ein deterministisches Verhalten. Wiedemann gibt für dieses Problem zwei Lösungsvorschläge an:

1. Die objektiven Größen DX, DV und $B(i-1)$ werden als Erwartungswerte betrachtet. Die für die Berechnung von $B(i)$ verwendeten Größen sind um diese Erwartungswerte normalverteilt.
2. Die Beschleunigung $B(i)$ wird mit den objektiven Größen DX, DV und $B(i-1)$ bestimmt, wobei dann das so errechnete $B(i)$ als normalverteilte Größe angesehenen wird.

Die Behandlung der Reaktionszeit hat Wiedemann in seinen Ausführungen speziell auf den von ihm gewählten Simulationszeitschritt von einer Sekunde ausgerichtet. Die nachfolgenden Ausführungen beziehen sich auf diese Zeitspanne und darauf, ab wann die ermittelte Beschleunigung $B(i)$ wirksam wird.

Aufgrund der Behandlung der Zeit als diskrete Größe, vergeht zwischen dem eigentlichen Überschreiten der Wahrnehmungsschwelle und der Feststellung dieses Ereignisses fast

ausnahmslos eine gewisse Zeit. Diese liegt zwischen null Sekunden und einer Sekunde. Auf Grund der Gleichverteilung jeder beliebigen Zeitspanne innerhalb dieses Intervalls, kann als modellbedingte Reaktionszeit 0.5 Sekunden erwartet werden. Die Reaktionszeit eines Fahrers liegt jedoch zwischen 0.5 Sekunden und 2 Sekunden. Für die Realisierung dieser Reaktionsdauern wird die $(0.5,0.15)$-normalverteilte Zufallsgröße z_n verwendet. Sie steht für den restlichen Anteil der gesamten Reaktionsdauer. Der Ausdruck $(1-z_n)$ repräsentiert dann die Zeitspanne innerhalb des Simulationszeitschritts von einer Sekunde, ab der die ermittelte Größe $B(i)$ wirksam sein soll. Da $B(i)$ aber innerhalb eines Simulationszeitschritts unveränderbar ist, muss es herabgemindert werden. Dies muss so geschehen, dass nach dem vollen Simulationszeitschrittintervall die gleiche Wirkung erzielt wird, wie wenn das nicht herabgeminderte $B(i)$ erst nach $(1-z_n)$ Sekunden wirksam würde. Um dies zu erreichen, wird $B(i)$ mit $(1-z_n)$ multipliziert.

Wiedemann berücksichtigt die Reaktionszeit in BREMSAX und BREMSBX nur dann, wenn die Bremslichter des Fahrzeuges $i-1$ nicht leuchten und $R(i)<2$ ist, also nicht schon seit längerem abgebremst wird.

2.3.3.4 Erweitertes Wiedemann-Modell

Harding stellt in seiner Dissertation [Hard07] eine Erweiterung des Wiedemann-Modells um ein Trödelverhalten der FahrerFahrzeuge vor. Ein FahrerFahrzeug fährt dabei für eine bestimmte Zeitdauer nicht seine Wunschgeschwindigkeit oder strebt diese an. Dadurch werden bei hohen Verkehrsstärken Stop-and-Go Wellen erzeugt, die im ursprünglichen Wiedemann-Modell nicht vorkommen. Die FahrerFahrzeuge führen mit einer gewissen Wahrscheinlichkeit eine festgelegte Zeitdauer lang nur noch Beschleunigungen von $-B_{null}$ aus. Der Trödelvorgang wird unterbrochen, wenn das Wiedemann-Modell eine größere Verzögerung als $-B_{null}$ ermittelt. Es wird angenommen, dass nur PKW trödeln, da Lastkraftwagenfahrern eine bessere Kontrolle über ihr Fahrzeug zugeschrieben wird. Ein FahrerFahrzeug i verfällt genau dann für die Zeitdauer $t_{tröd}$ ins trödeln, wenn eine gleichverteilte Zufallszahl in den Grenzen von 0 bis 1 größer ist als die Trödelwahrscheinlichkeit $z_{tröd}(i)$, deren Wertebereich ebenfalls in den Grenzen von 0 bis 1 verläuft. Eine Trödelwahrscheinlichkeit von 1 bedeutet, dass das FahrerFahrzeug nie trödelt und 0, dass es immer trödelt. Ferner wird angenommen, dass ein FahrerFahrzeug nur dann trödelt, wenn es sich im Zustand WUNSCH oder FOLGEN befindet.

Ein weiterer Schwachpunkt im ursprünglichen Wiedemann-Modell ist, dass für alle FahrerFahrzeuge gleiche $B_{\min}$ und $B_{\max}$ angenommen werden. Deswegen soll für ein FahrerFahrzeug i vom Typ PKW gelten:

$$B_{\max,PKW}(i) = (0.2 + 0.8 \cdot z_{bw}(i)) \cdot (8 - \sqrt{v(i)}) \quad [m/s^2], \tag{2.43}$$

$$B_{\min,PKW}(i) = -8 - 2 \cdot z_{bw}(i) + 0.036 \cdot v(i) \quad [m/s^2]. \tag{2.44}$$

Für ein FahrerFahrzeug i vom Typ LKW soll gelten:

$$B_{\max,LKW}(i) = 1.581 - 0.057 \cdot v(i) + (0.6 - 0.01 \cdot v(i)) \cdot z_{bw}(i) \quad [m/s^2], \tag{2.45}$$

$$B_{\min,LKW}(i) = -6 - 2 \cdot z_{bw}(i) + 0.9 \cdot v(i) \quad [m/s^2].^{24} \tag{2.46}$$

2.3.4 Spurwechselmodelle

2.3.4.1 Überblick

Die Ausführungen zu den Spurwechselmodellen basieren auf [Erle07].
Für die realitätsnahe mikroskopische Simulation von Verkehr auf mehrspurigen Straßen ist es wichtig, dass die einzelnen FahrerFahrzeuge Spurwechsel vornehmen können. Wobei dies insbesondere für Straßen mit mehr als einer Richtungsfahrbahn zutrifft. Bei zweispurigen Straßen mit jeweils einer Richtungsfahrbahn spielen Überholvorgänge (also Spurwechsel) keine so große Rolle. Überholmöglichkeiten sind bei den allgemein eher hohen Verkehrsdichten auf den Straßen selten gegeben. Die existierenden Spurwechselmodelle konzentrieren sich daher auf die sehr komplexen strategisch-kooperativen Spurwechselvorgänge auf Strecken mit mehreren Richtungsfahrbahnen, da sie hier für die Verkehrsverteilung auf die einzelnen Spuren von großer Bedeutung sind. Die weiteren Ausführungen im gesamten Abschnitt 2.3.4 beziehen sich auf diese Situation.
Die Abwägungs- und Entscheidungsprozesse darüber, wann, wie und ob überhaupt ein Spurwechsel durchgeführt werden soll, sind sehr vielfältig und komplex. Bei den im Abschnitt 2.3.3 besprochenen mikroskopischen Modellen hing die Entscheidung über die vorzunehmende Beschleunigung zum größten Teil vom direkt vorausfahrenden FahrerFahrzeug ab. Bei der Entscheidung für einen Spurwechsel sind in der Regel mehrere FahrerFahrzeuge zu beachten. Die Motivationen eines Fahrers für einen Spurwechsel sind im Wesentlichen:

- Der Fahrer möchte ein langsameres Fahrzeug überholen.
- Der Fahrer möchte einem auffahrenden Fahrzeug die Zufahrt erleichtern.
- Der Fahrer möchte selbst auf eine Strecke auffahren.
- Der Fahrer möchte von einer Strecke abfahren.

[24] Die Gleichungen (2.43) bis (2.46) sind aus [Erle07] entnommen.

Sie sind auch kombiniert denkbar, zum Beispiel möchte der Fahrer ein langsameres Fahrzeug überholen und gleichzeitig einem auffahrenden Fahrzeug die Einfahrt erleichtern. Grundsätzlich ist der Fahrer dabei bestrebt, Gefahrensituationen und Behinderungen der anderen Verkehrsteilnehmer zu vermeiden.
Bei der Entscheidungsfindung ist also der gesamte situative Kontext zu betrachten und auszuwerten. Es gelingt aber nur bedingt, das ganze Spektrum der unterschiedlichen Fahrsituationen realitätsgetreu zu simulieren. Einige vorhandene Spurwechselmodelle konzentrieren sich daher auch nur auf die Nachbildung von bestimmten Fahrsituationen, wie Überholmanöver oder das Auffahren auf eine Strecke. In den nächsten beiden Abschnitten sollen zwei Spurwechselmodelle kurz vorgestellt werden: das Sparmann-Modell und das Theis-Modell (veröffentlicht in [Spar78] und in [Theis97]).

2.3.4.2 Sparmann-Modell

Sparmann entwickelte sein Modell aufbauend auf dem in Abschnitt 2.3.3.3 beschriebenen Wiedemann-Modell. FahrerFahrzeuge können sich vor und nach einem Spurwechsel in einem von drei Zuständen wiederfinden, die sich an die des Wiedemann-Modells anlehnen. Diese drei Zustände sind:

- **Unbeeinflusst.** Der Fahrer wird nicht durch Nachbarfahrzeuge beeinflusst.
- **Potenziell beeinflusst.** Der Fahrer ist gegenwärtig nicht durch andere Fahrzeuge beeinflusst, wird dies aber wahrscheinlich in naher Zukunft sein.
- **Aktuell beeinflusst.** Der Fahrer wird gegenwärtig unbewusst, bewusst oder kritisch beeinflusst.

Bevor ein Fahrzeug die Spur wechseln kann, muss bei dessen Fahrer zunächst ein Bedürfnis danach entstehen. Damit ein Fahrer einen Spurwechselwunsch verspürt, müssen zwei Regelsätze erfüllt sein, die von Sparmann definiert wurden. Dabei wird zwischen einem Spurwechselwunsch nach links oder nach rechts unterschieden.[25] Dieser baut sich über einen bestimmten Zeitraum auf und wird immer größer, was die Wahrscheinlichkeit für einen tatsächlichen Wechsel erhöht. Das schiere Vorhandensein dieses Wunsches führt jedoch nur dann zur Entscheidung für den Spurwechsel, wenn gewisse Sicherheitsbedingungen erfüllt sind. Es dürfen keine Gefährdungen des Überholenden oder der in seiner Nähe befindlichen Verkehrsteilnehmer entstehen und diese auch möglichst nicht behindert werden.
Eine beispielhafte Situation für das Wechseln eines FahrerFahrzeuges auf die linke Spur ist dann gegeben, wenn das vorausfahrende FahrerFahrzeug deutlich langsamer als mit der eigenen angestrebten Wunschgeschwindigkeit fährt. Das FahrerFahrzeug mit dem Überholwunsch wird allerdings nur dann den Spurwechsel durchführen, wenn das auf der linken Spur hinter ihm fahrende FahrerFahrzeug dadurch entweder gar nicht oder nur potenziell beeinflusst wird. Verschwindet der Grund für den Spurwechselwunsch, baut sich dieser wieder über einen gewissen Zeitraum ab. Für den Auf- und Abbau des Spurwechselwunsches und über die

[25] Eine vollständige Auflistung aller zu berücksichtigenden Situationen ist in [Spar78] zu finden.

Entscheidung darüber werden die Geschwindigkeiten, Abstände und angestrebten Sicherheitsabstände der interagierenden FahrerFahrzeuge bestimmt.[26] Auf deren Grundlage wird dann über Vergleiche entschieden, ob sich ein FahrerFahrzeug im Bezug zu einem anderen in einem unbeeinflussten, potenziell beeinflussten oder aktuell beeinflussten Zustand befindet.
Das Sparmann-Modell betrachtet nur direkt benachbarte FahrerFahrzeuge. Dadurch werden andere potenzielle Interaktionspartner nicht mit berücksichtigt, zum Beispiel ein auffahrendes FahrerFahrzeug an einer Anschlussstelle. Im Sparmann-Modell ist auch kein kooperativer Ansatz vorgesehen, um solchen FahrerFahrzeugen das Auffahren durch einen Spurwechsel zu erleichtern. Die Fahrer handeln nur aus egoistischen Motiven heraus. Obwohl die Spurwechselvorgänge aus der mikroskopischen Betrachtung heraus realistisch wirken, ergibt sich makroskopisch eine zu starke Verkehrsverteilung auf die rechte Spur.

2.3.4.3 Theis-Modell

Theis‘ Modell basiert auf den Arbeiten von Wiedemann und Sparmann. Das beschriebene Spurwechselverhalten bezieht sich jedoch nur auf planfreie Anschlussstellen, wie Autobahnauffahrten. In diesem Spurwechselmodell ist kooperatives Verhalten der FahrerFahrzeuge umgesetzt, indem diese andere FahrerFahrzeuge um ihre Unterstützung bitten können. Auch im Modell von Theis kann sich ein Fahrer in drei verschiedenen Zustandsarten befinden:

- **Einfahrer.** Damit werden FahrerFahrzeuge bezeichnet, die an einem Verflechtungsbereich oder einer Einfahrt auf die linke Spur wechseln, um auf ihre Wunschspur zu gelangen.
- **Ausfahrer.** Damit werden FahrerFahrzeuge bezeichnet, die am nächsten Verflechtungsbereich oder der nächsten Ausfahrt auf die rechte Spur wechseln.
- **Durchfahrer.** Damit werden FahrerFahrzeuge bezeichnet, die auf ihrer gegenwärtigen Spur weiter fahren können, aber dabei andere Ein- und Ausfahrer nach Möglichkeit unterstützen.

Sowohl ein Ausfahrer auf der linken Spur der Hauptfahrbahn, als auch ein Einfahrer wollen auf die rechte Spur der Hauptfahrbahn wechseln. Dafür suchen sie sich die nächstliegende Lücke in Fahrtrichtung. Rückwärtige Lücken werden nicht betrachtet, da die Fahrzeuge mit Spurwechselwunsch hierzu verzögern müssten, was die durchschnittliche Geschwindigkeit senken würde. Um in eine erkannte Lücke zu wechseln, beschleunigt das FahrerFahrzeug bis es genau neben der Lücke fährt. Anschließend passt es seine Geschwindigkeit an die seines neuen Vordermanns an. Jetzt erfolgt der Wechsel auf die andere Spur, wobei das einscherende FahrerFahrzeug die entsprechenden Sicherheitsabstände[27] beachtet. Ist die Lücke, in die gewechselt werden soll, jedoch zu klein, wird das FahrerFahrzeug, welches die hintere Begrenzung der Lücke bildet, um Unterstützung gebeten. Dieses verlangsamt daraufhin, beziehungsweise wechselt bei einem Einfahrvorgang die Spur. Das rücksichtnehmende

[26] Das geschieht nach dem Prinzip des Wiedemann-Modells aus Abschnitt 2.3.3.3.
[27] Die Sicherheitsabstände werden nach den Prinzipen des Wiedemann-Modells ermittelt.

FahrerFahrzeug muss solange das wechselnde unterstützen, bis jenes die Spur gewechselt oder ein anderes FahrerFahrzeug um Hilfe gebeten hat. Ein FahrerFahrzeug kann immer nur von höchstens einem anderen unterstützt werden.
Ein einfahrendes FahrerFahrzeug sucht bereits auf der Auffahrtrampe durch Projektion seiner Position auf die Hauptstrecke nach einer geeigneten Lücke. Sobald das FahrerFahrzeug dann den Einfahrstreifen erreicht hat, passt es seine Position und Geschwindigkeit so an, dass es vor dem Ende des Einfahrstreifens die angestrebte Lücke erreicht hat. Dabei ist das FahrerFahrzeug bereit, ein höheres Risiko in Kauf zu nehmen je näher es dem Ende seiner Spur kommt. Es akzeptiert jetzt immer kleinere Sicherheitsabstände bis hin zu seinem Stillstands-Sicherheitsabstand AX, der aus der Beschreibung des Wiedemann-Models in Abschnitt 2.3.3.3 bekannt ist. Findet ein FahrerFahrzeug trotz des Akzeptierens kleinerer Sicherheitsabstände und der zusätzlichen Unterstützung durch ein anderes FahrerFahrzeug keine geeignete Lücke, dann bremst es kurz vor dem Ende seines Einfahrtstreifens ab und wartet auf einen geeignete Lücke, in die es wechseln kann. Konnte ein Fahrer nur in eine Lücke wechseln, weil er den eigentlich von ihm gewünschten Sicherheitsabstand unterschritten hatte, ist er nun bestrebt, diesen wieder zu erreichen. Dabei ist es wichtig, dass der Fahrer dies nicht allzu schnell durchführt. Durch Bremsen kann es auf Grund der Rückkopplungseffekte sonst zu einem Stau oder Stop-and-Go kommen.
Nicht jedes FahrerFahrzeug kann dabei zu jeder Zeit jede Aktion durchführen. Theis definiert für jeden der drei beschriebenen Zustände, unterteilt nach bestimmten Abständen zu einer Ausfahrt, die zulässigen Aktionen.[28] So wird zum Beispiel ein FahrerFahrzeug 500 Meter vor seiner Auffahrt nicht noch einmal auf die linke Spur wechseln.
Das Theis-Modell ist sehr komplex und detailliert. Für jede Situation wird eine genaue Verhaltensweise vorgegeben. Es ist jedoch zu beachten, dass dieses Modell nur Spurwechselverhalten an Anschlussstellen beschreibt. Das Überholen zwischen den Anschlussstellen zum Zwecke der Geschwindigkeitssteigerung wird nicht erfasst. Bei hohen Verkehrsstärken kommt es im Auffahrtsbereich auf der rechten Spur zu Verlangsamungen, die bis hin zu einem kompletten Erliegen des Verkehrsflusses auf dieser Spur führen können. Das ist durch die Verzögerungen der FahrerFahrzeuge auf der rechten Spur begründet, die es anderen erleichtern wollen aufzufahren.[29]

2.3.5 Mesoskopische Verkehrsmodelle

Mesoskopische Verkehrsmodelle stellen eine Verbindung zwischen den im Abschnitt 2.3.2 beschriebenen makroskopischen Verkehrsmodellen und den im Abschnitt 2.3.3 beschriebenen mikroskopischen Verkehrsmodellen dar. Dadurch wird versucht, die Vorteile der makroskopischen mit denen der mikroskopischen Betrachtungsweise zu verbinden. Zum Beispiel können

[28] Die Abstände sind 1000 Meter und 500 Meter vor Ausfahrtsbeginn sowie dem Ausfahrtsverflechtungsbereich.
[29] In [Erle07] wird ein Ansatz beschrieben, mit dem sich dieses Problem verringern lässt.

die Knotenpunkte in einer Verkehrssimulation durch mikroskopische Modelle simuliert werden, während auf dem restlichen Streckennetz makroskopische Modelle als Basis für die Simulation dienen. Es existieren Verkehrssimulationsprogramme, die eine solche direkte Kopplung ermöglichen.[30]

Zudem gibt es auch Modelle, welche die mikroskopische mit der makroskopischen Betrachtungsweise verbinden. Zu nennen wäre hier das Modell, das von Prigogines und seinen Mitarbeitern entwickelt wurde (beschrieben in [PrigHerm71]). Es ist das erste mesoskopische Verkehrsmodell und ist von der Boltzmann'schen kinetischen Gastheorie (beschrieben in [Bolt95]) beeinflusst. Ein näheres Eingehen auf dieses Modell ist an dieser Stelle jedoch nicht vorgesehen.

2.3.6 Zusammenfassendes Fazit

Grundsätzlich ist jedes Modell auf irgendeine Art oder Weise unvollständig oder fehlerhaft, da es immer nur einen Ausschnitt der Wirklichkeit erklärt und nicht die gesamte Realität abbildet. Jedes Modell hat somit immer seine Vor- und Nachteile und ist daher für unterschiedliche Betrachtungen mehr oder weniger geeignet.

Die drei Arten von Verkehrsmodellen (makroskopische, mikroskopische, mesoskopische) unterscheiden sich hauptsächlich in der Betrachtungsweise der einzelnen FahrerFahrzeuge und in dem erforderlichen Rechenaufwand. Makroskopische Verkehrsmodelle werden für die Simulation von großen Streckennetzen eingesetzt. Ihr größter Vorteil besteht darin, dass der Rechenaufwand unabhängig von der Anzahl der FahrerFahrzeuge ist, da diese nicht einzeln betrachtet werden. Deswegen können auch nur makroskopische Verkehrskenngrößen, wie die Verkehrsdichte oder die Verkehrsstärke, beobachtet und ausgewertet werden. Diese makroskopische Art der Betrachtungsweise ist für bestimmte Einsatzzwecke auch völlig ausreichend. Besteht lediglich Interesse daran, die Auswirkungen von geplanten Straßenbauprojekten auf das System einer Stadt oder ein überregionales Gebiet zu erfahren, kann das sehr gut mit makroskopischen Modellen geschehen. Es gibt eine Reihe von Verkehrssimulationssoftware, die auf Basis makroskopischer Modelle Verkehr simuliert (zum Beispiel VISUM). Eher ungeeignet sind makroskopische Modelle für die Betrachtung von Knotenpunkten und Engstellen, da hier das individuelle Verhalten und die gegenseitige Beeinflussung der spezifischen Fahrer-Fahrzeuge von großer Bedeutung sind.

Mikroskopische Modelle zeichnen sich dadurch aus, dass jedes einzelne FahrerFahrzeug simuliert wird. Dadurch steigt der Rechenaufwand mit der Anzahl an simulierten FahrerFahrzeugen an. Wie groß das simulierte Streckennetz ist, hängt also maßgeblich von der zur Verfügung stehenden Rechenleistung ab. Besonders vorteilhaft bei mikroskopischen Verkehrsmodellen ist, dass sowohl spezifische Kenngrößen der FahrerFahrzeuge als auch makroskopische Kenngrößen, wie die Verkehrsdichte und –stärke, betrachtet werden können. Aufgrund der Modellierung der einzelnen FahrerFahrzeuge lassen sich Knotenpunkte und

[30] Ein Beispiel sind die Software VISUM und VISSIM. Siehe dazu auch Abschnitt 2.4.4.2.

Engstellen gut simulieren. Sofern das konkret eingesetzte mikroskopische Modell dies zur Verfügung stellt, kann auch der Einfluss unterschiedlicher Verhaltensweisen von FahrerFahrzeugen und insbesondere deren Auswirkung auf andere untersucht werden. Zwingend ist der Einsatz von mikroskopischen Modellen, wenn der simulierte Verkehr zudem visuell realistisch dargestellt werden soll.
Dabei ist ein wirkliches realistisches Nachbilden von Spurwechselvorgängen in sämtlichen Situationen ein äußert schwieriges und komplexes Unterfangen. Durch die Kombination des Sparmann- mit dem Theis-Modell ließe sich das trotz der erläuterten Probleme lösen. Die Kombination beider Modelle könnte wie folgt geschehen: Auf den Streckenabschnitten zwischen den Anschlussstellen wird das Sparmann-Modell verwendet und ab einer bestimmten Stelle vor der Anschlussstelle bis zu einer gewissen Stelle danach das Theis-Modell.
Die mesoskopischen Modelle sind im praktischen Sinne vor allem dann interessant, wenn sie als Kopplung von makroskopischen Modellen und mikroskopischen Modellen in einem Softwaresystem verwendet werden. In theoretischer Hinsicht sind sie von besonderem Interesse, um aus der mikroskopischen Ebene konsistent die makroskopische Betrachtungsebene herzuleiten.

2.4 Verkehrssimulation

2.4.1 Einführung

Simulation im Allgemeinen „ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell“ (aus [VDI96]). Dabei wird versucht, Erkenntnisse, die aus der Untersuchung des Modells gewonnen wurden, auf das reale System zu übertragen.
Als Verkehrssimulation soll die Abbildung des Verkehrs mit seinen Verkehrsteilnehmern und deren Verhalten sowie die des Streckennetzes mittels eines Softwaresystems verstanden werden. Wie und welche Verkehrsteilnehmer nachzubilden sind, ist von dem zu untersuchenden Problemgebiet abhängig. Grundsätzlich existieren Softwaresysteme für die Simulation aller wichtigen Verkehrsträger, wie Straßenverkehr, Schifffahrtsverkehr, Luftverkehr und Fußgängerverkehr. Für jeden dieser Verkehrsträger gibt es verschiedene Modelle und Randbedingungen, um ihr Verhalten möglichst realistisch nachzubilden. Es können verschiedene Verkehrsträger und Fahrzeugtypen in einer Simulation zusammengefasst werden, beispielsweise der innerstädtischen Verkehr samt Fahrzeugen, Bussen, U- und S-Bahnen sowie Fußgänger. So entstehen äußerst komplexe Simulationen, bei denen die verschiedenen Verhaltensmodelle integriert werden müssen. Damit eine wirklich realitätsnahe Simulation von Verkehr durchgeführt werden kann, müssen seine Gesetzmäßigkeiten verstanden sein. Nur so kann es gelingen, dass sowohl das Verhalten der einzelnen Fahrzeuge sowie das des Gesamtverkehrssystems die Realität abbilden.

2.4.2 Detaillierungsgrad

Wenn ein Simulationsmodell entwickelt wird, ist es von wesentlicher Bedeutung wie detailliert die Nachbildung der Realität im Modell erfolgt. Im Allgemeinen führt eine präzise und ausführliche Beschreibung des Verhaltens der Fahrzeuge zu einem rechenintensiven Modell mit einem hohen Grad an Kalibrierungsarbeit. Entsprechend klein ist dann der Realitätsausschnitt, der simuliert werden kann. Wenn aber eine wesentlich undetailliertere Nachbildung der Fahrzeuge und ihres Verhaltens vorgenommen wird, können entsprechend größere Realitätsausschnitte untersucht werden. Beides hat, je nach Untersuchungsgegenstand, seine Berechtigung. So werden bei den Verkehrsmodellen die makroskopische, die mesoskopische und die mikroskopische Betrachtungsebene unterschieden.[31]

2.4.3 Zeitverhalten

Bei der computergestützten Simulation wird zwischen der Simulationszeit und der Realzeit unterschieden. Mit Simulationszeit ist der Zeitverlauf innerhalb der Simulation gemeint. Die Realzeit hingegen entspricht dem vom Menschen empfundenen Zeitverlauf. Beide Zeitverläufe sind oftmals nicht synchron. Ein Beispiel: Der Computer berechnet für ein simuliertes Fahrzeug den Weg, den es bei einer Geschwindigkeit von 10 m/s innerhalb von einer Sekunde (Simulationszeit) zurücklegt. Für diese Berechnung benötigt der Computer aber nur eine Millisekunde (Realzeit). Das bedeutet, dass der Computer innerhalb einer real verstreichenden Sekunde den Wegverlauf des Fahrzeuges für 1000 Sekunden Simulationszeit berechnen kann. Es ist jedoch nicht zwingend, dass die Realzeit deutlich langsamer verläuft als die Simulationszeit. Es wäre genauso gut denkbar, dass der Computer für die Berechnung einer Sekunde Simulationszeit eine Minute Realzeit benötigt. Würde der Bewegungsverlauf des Fahrzeuges während der Berechnungsdauer visualisiert werden, ergäbe sich für den menschlichen Betrachter bei der ersten Variante (Simulationszeit > Realzeit) ein zeitgeraffter Bewegungsablauf, bei der zweiten Variante (Simulationszeit < Realzeit) hingegen ein zeitlupenartiger Bewegungsablauf.
Die folgenden Definitionen beziehen sich auf [Kope97].
Sind bei einer Simulation Simulationszeit und Realzeit synchron zueinander, wird von einer Echtzeitsimulation gesprochen. Wobei das Charakteristikum „Echtzeit“ keine Aussage über die Geschwindigkeit der Berechnung darstellt (vergleiche [Kope97]). Es bedeutet lediglich, dass eine Berechnung innerhalb einer definierten Zeitdauer fertiggestellt wird. Ist das Berechnungsergebnis gültig, trifft aber zu spät ein, ist dies als Fehler zu interpretieren. Üblicherweise wird noch einmal zwischen „weicher Echtzeit“ und „harter Echtzeit“ unterschieden. Dabei werden folgende Anforderungen gestellt:

- **Weiche Echtzeit** Anforderungen bedeuten, dass das Berechnungsergebnis in der Regel innerhalb des definierten Zeitintervalls eintrifft. Eine Überschreitung des Zeitin-

[31] Eine Erläuterung der verschiedenen Betrachtungsebenen erfolgt im Abschnitt 2.3, wo ausführlich auf die mikroskopische Ebene eingegangen wird.

tervalls ist aber nicht auszuschließen. Dies ist zwar ein Fehler, führt jedoch im Allgemeinen nicht zum Abbruch. Ein Beispiel für eine weiche Echtzeit Anwendung sind Audio- und Videostreaming. In der Regel arbeitet die Anwendung so schnell, dass eine flüssige Ton- und Bildwiedergabe gegeben sind. Wird das Zeitintervall jedoch einmal überschritten, kommt es zu Bildruckeln beziehungsweise Tonaussetzern. Solche kurzen Störungen stellen jedoch keinen Grund dar, das Streaming komplett abzubrechen.

- **Harte Echtzeit** Anforderungen verlangen, dass das Berechnungsergebnis garantiert innerhalb des definierten Zeitintervalls eintrifft und sich dieses auch beweisen lässt. Eine Überschreitung des Zeitintervalls ist ein schwerer Fehler, der einer besonderen Behandlung bedarf. Wenn zum Beispiel die Berechnungen für die Motorsteuerung eines Fahrzeugs nicht rechtzeitig abgeschlossen sind, kann es zu einem Stottern oder zum Ausfall des Motors kommen.

Außerdem gilt es zu beachten, dass ein Computer immer nur in diskreten Zeitschritten arbeiten kann, während die Zeit aber einen kontinuierlichen Verlauf hat. Der Verlauf einer zeitabhängigen Größe kann daher auch nur diskretisiert beobachtet werden. Das bedeutet, dass die Simulationszeit in Zeitschritte mit einer entsprechenden Dauer unterteilt ist. Grundsätzlich lässt sich für die Bestimmung der Zeitschrittdauer zwischen dem zeitschrittorientierten und dem ereignisorientierten Ansatz unterscheiden.

Beim zeitschrittorientierten Ansatz wird die Simulationszeit in Zeitschritte mit konstanter Dauer von zum Beispiel einer Sekunde unterteilt. Somit kann immer nur beobachtet werden, wie sich die zeitabhängigen Größen innerhalb einer Sekunde verändert haben.

Der ereignisorientierte Ansatz sieht dynamische Aktualisierungsraten vor. Eine Größe wird immer nur dann neu berechnet, wenn ein bestimmtes Ereignis eingetreten ist, zum Beispiel eine LSA auf Rot geschalten hat oder sich die Geschwindigkeit eines vorausfahrenden Fahrzeuges verändert hat. Der zeitschrittorientierte Ansatz findet am häufigsten Verwendung, weil er einfacher zu implementieren und vielfältiger einsetzbar ist.

2.4.4 Verkehrssimulationssoftware

2.4.4.1 Vorbemerkung

Es existiert eine Vielzahl von mikroskopischen Verkehrssimulationsprogrammen.[32] Obwohl es darunter viele kommerzielle gibt, können jedoch nur VISSIM, AIMSUN und PARAMICS eine breite Nutzerbasis in Europa vorweisen (vergleiche [Aim10], [Par10] und [Ptv10c]). Sie stellen umfangreiche leistungsstarke Komplettsysteme mit einem reichen Funktionsumfang und hohem Bedienkomfort dar. Im folgenden Abschnitt soll ein kurzer Überblick über die Funkti-

[32] Für eine Erläuterung der makroskopischen, mikroskopischen und mesoskopischen Betrachtungsebenen siehe die Abschnitte 2.3.2, 2.3.3 und 2.3.5.

onsumfänge von VISSIM und seines makroskopischen Pendants VISUM gegeben werden, da sie in Deutschland weit verbreitet sind.

2.4.4.2 VISSIM und VISUM

VISSIM und VISUM sind Programmsysteme der ptv AG aus Karlsruhe. Mit VISUM kann eine multimodale Verkehrsplanung von regionaler bis nationaler Ebene durchgeführt werden (siehe Abbildung 2.6). Individual- und öffentlicher Verkehr mit verschiedenen Verkehrsträgern und in verschiedenen Verkehrsnetzen können nachgebildet, simuliert und analysiert werden. Es entstehen komplexe Verkehrsnetze mit Straßen und Schienen sowie unterschiedlichen Verkehrsträgern wie PKW, LKW, Straßenbahnen, Bussen, Zügen, Fähren und anderen. Dabei müssen die verschiedenen Verhaltensmodelle integriert werden.
VISSIM ist ein leistungsstarkes mikroskopisches Softwaresimulationssystem. Es basiert auf den Arbeiten von Wiedemann (siehe Abschnitt 2.3.3.3) und simuliert das Verhalten der Fahrzeuge auf der Grundlage seines Modells. Dem Spurwechselverhalten liegt das Sparmann-Modell zugrunde (siehe Abschnitt 2.3.4.2). VISSIM ist speziell für die Simulation von innerstädtischem Verkehr ausgelegt, kann aber auch für Autobahnverkehr genutzt werden. Durch VISSIM ist die Simulation des komplexen städtischen Verkehrssystems mit PKW, ÖPNV-Fahrzeugen, Fahrradfahrern und Fußgängern möglich.

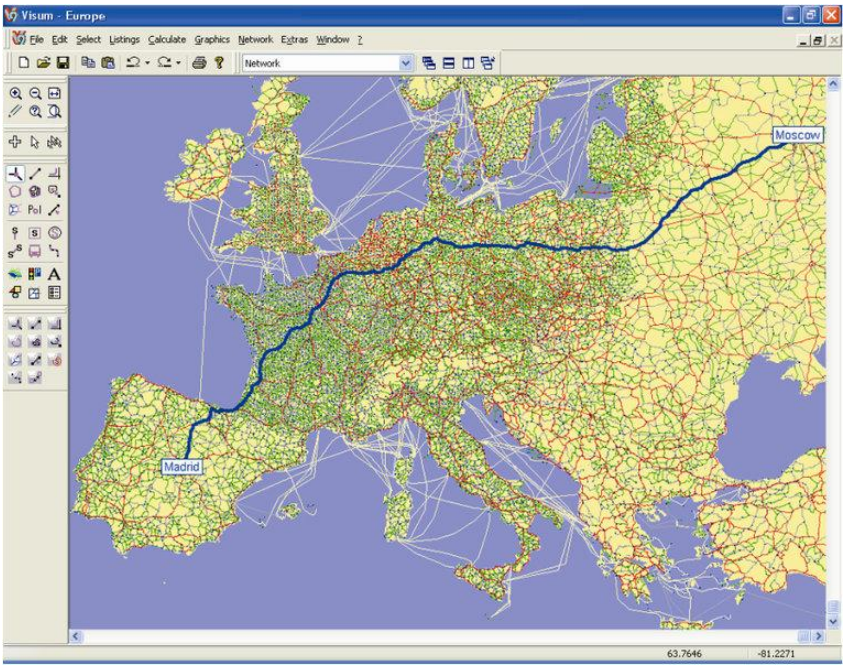

Abbildung 2.6: Europäisches Verkehrsnetz in VISUM (Entnommen von [Ptv10a])

In den Softwaresystemen reagieren die Verkehrsteilnehmer auf Verkehrszeichen, und es ist möglich, LSA-Steuerungsprogramme einzugeben. Die Darstellungen des Verkehrsnetzes und der Verkehrsteilnehmer können in einer 2D-Entwurfs- und Simulationsansicht (siehe Abbildung 2.7 links, Seite 37) und in einer 3D-Simulationsansicht (siehe Abbildung 2.7 rechts, Seite 37)

erfolgen. Es besteht die Möglichkeit, anhand eines Satellitenbildes ein Straßennetz per Point-and-Click abzufassen und die Darstellungen zu überlagern.

Abbildung 2.7: 2D- und 3D-Ansicht in VISSIM (Entnommen von [TUBe10] (links) und [Ptv10b] (rechts))

VISSIM und VISUM können zu einer mesoskopischen Betrachtungsweise zusammengeführt werden, indem die Simulation bestimmter Knotenpunkte mit VISSIM mikroskopisch und das restliche Verkehrsnetz mit VISUM makroskopisch erfolgt. Sowohl auf VISUM als auch auf VISSIM kann über eine COM-Schnittstelle softwaretechnisch zugegriffen werden.

2.4.5 Fahrzeugsimulationssoftware

2.4.5.1 Vorbemerkung

Im folgenden Abschnitt soll die Fahrzeugsimulationssoftware CarMaker der IPG Automotiv GmbH aus Karlsruhe vorgestellt werden, da das Verkehrssimulationsmodul indirekt mit ihm zusammenarbeitet. Es ist außerdem beabsichtigt, die in CarMaker vorhandenen Funktionalitäten für die Verkehrsabbildung zu nutzen. Die berechneten Daten des Verkehrssimulationsmoduls sollen die Steuerung des Verkehrs in CarMaker übernehmen. Das geschieht allerdings nicht im Rahmen dieser Masterarbeit.

2.4.5.2 CarMaker

CarMaker bildet das physikalische Modell eines Fahrzeuges umfassend ab und berücksichtigt bei simulierten Fahrten auch äußere auf das Fahrzeug einwirkende Kräfte, wie den Abrollwiderstand der Reifen und den Luftwiderstand. CarMaker verfügt zudem über ein benutzerfreundliches User Interface. Fahrzeuge sind von den Reifen über die Masseverteilung bis zu den Bremsen und dem Getriebe umfassend parametrisierbar (siehe Abbildung 2.8 links, Seite 38). Auch Sensoren und Assistenzsysteme lassen sich nachbilden und ihr Verhalten simulieren. Auf unterschiedlichen, hinzuladbaren Strecken kann das Fahrzeug fahren und dabei von einer

KI[33] gelenkt werden oder vorher definierte Fahrmanöver ausführen. In der Softwarekomponente IPGMovie kann die Fahrt des Fahrzeuges aus unterschiedlichen Kameraperspektiven betrachtet werden (siehe Abbildung 2.8 rechts).

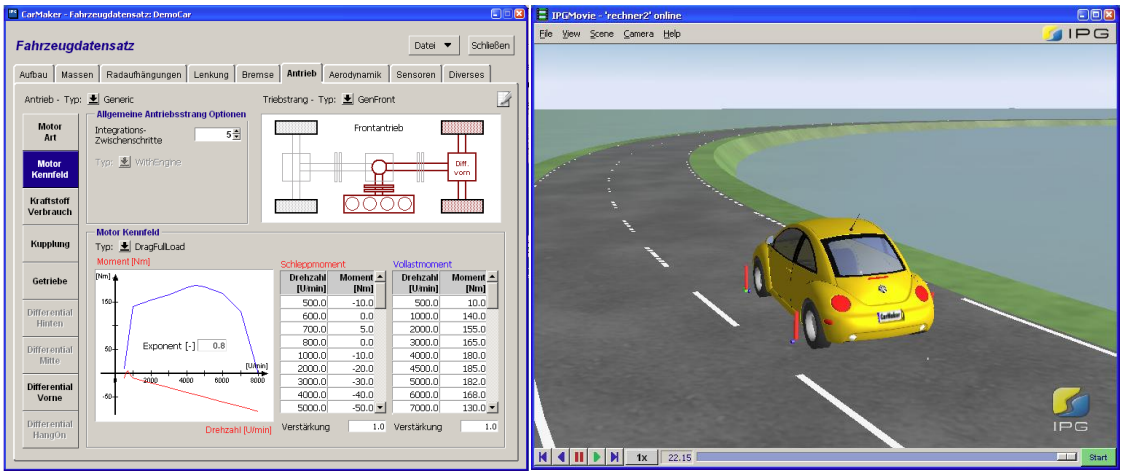

Abbildung 2.8: Parametereinstellung des Antriebs und 3D Darstellung

Die Simulationszeit kann gerafft, in Echtzeit oder verlangsamt ablaufen. Es besteht die Möglichkeit, hunderte verschiedene Größen[34], wie die Geschwindigkeit oder die Beschleunigung, mit der Softwarekomponente IPGControl zu protokollieren und ihren Verlauf während der Simulation grafisch darzustellen (siehe Abbildung 2.9). Die Darstellung der Größen kann dabei nicht nur über die Zeit erfolgen, sondern auch in beliebiger Relation zueinander.

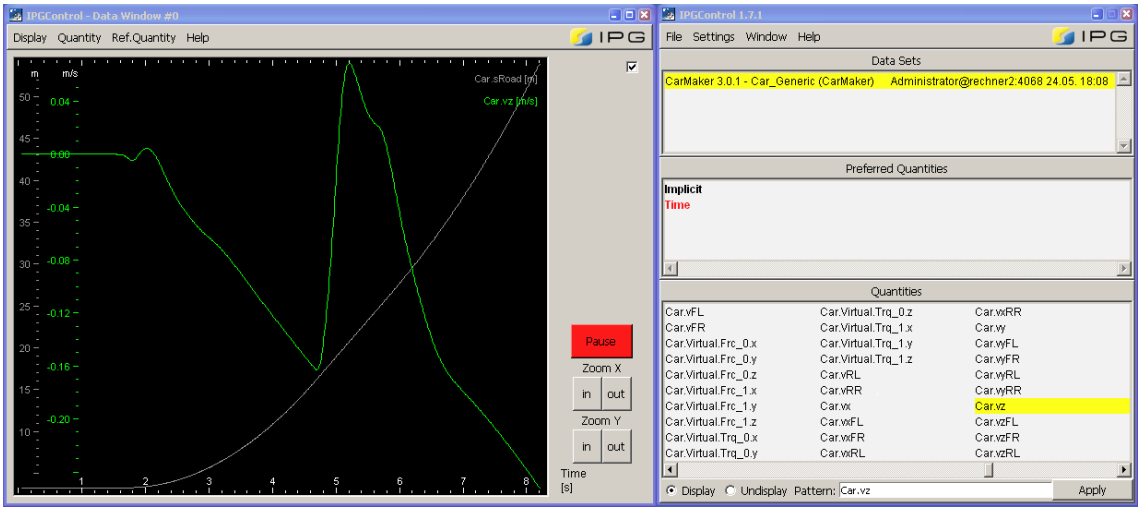

Abbildung 2.9: Darstellung von Größen in IPG Control

Verkehr ist in CarMaker gegenwärtig nur äußerst rudimentär darstellbar. Die Fahrzeuge sind physisch durch Quader repräsentiert, denen ein Fahrzeugtyp zugewiesen werden kann. Die

[33] Auch die KI ist umfassend parametrisierbar und kann verschiedene Fahrertypen nachbilden.
[34] Solch eine Größe wird in CarMaker als *Quantity* bezeichnet.

Bewegung der Fahrzeuge ist nur über die Definition von Fahrmanövern möglich. Ein Verkehrsmodell ist nicht integriert.
CarMaker erlaubt einen vielfältigen Zu- und Eingriff. Über TCL-Scripte können zum Beispiel verschieden Simulationsläufe automatisiert geladen, parametrisiert und abgearbeitet werden. Zudem lassen sich sämtliche Quantitys übersteuern. Dies kann eingeschränkt über die GUI oder mit einem C-Programm durchgeführt werden. Weiterhin kann an wohldefinierten Punkten direkt programmatisch in die Simulationsschleife eingegriffen werden.

3 Modellierung des Verkehrssimulationsmoduls

3.1 Vorbemerkungen

In diesem Kapitel soll die Modellierung des Verkehrssimulationsmoduls erläutert werden. Es steht im engen Zusammenhang mit Kapitel 4, in dem dessen Implementierung beschrieben wird. Ausgehend von einem ersten Modellentwurf wurde mit der Implementierung des Moduls begonnen. Daraus gewonnene Erkenntnisse flossen bei der Überarbeitung und Weiterentwicklung des Modellentwurfs mit ein. Am Ende dieses andauernden, mehrstufigen Entwicklungsprozesses steht das in diesem Kapitel vorgestellte Modell. Währenddessen wurden einige durchaus kritisch zu betrachtende Modellierungsentscheidungen getroffen. An den entsprechenden Stellen im Text sind diese erläutert und die getroffenen Entscheidungen ausführlich begründet.

3.2 Anforderungen

Das Verkehrssimulationsmodul muss eine Reihe von anspruchsvollen Anforderungen erfüllen. Zunächst werden die allgemeinen modellierungstechnischen beschrieben, bevor näher auf die speziellen und informationstechnisch anspruchsvollen Anforderungen eingegangen wird. Das Verkehrssimulationsmodul soll mit dem objektorientierten Paradigma modelliert und programmiert werden. Die Vorteile des objektorientierten Ansatzes sind allgemein bekannt, weswegen auch keine weiteren Begründungen für dessen Wahl erfolgen sollen. Außerdem werden beim Entwurf des Objektmodells eine möglichst lose Kopplung und eine hohe Kohärenz angestrebt, damit das Modul gut wartbar und erweiterbar ist. Für den Entwurf sollen die Methoden und Techniken der UML angewendet werden.[35] Die Bezeichner im Entwurf und im Programmcode sind in Englisch.

Das Verkehrssimulationsmodul soll realistischen Verkehr auf der Basis von mikroskopischen Verkehrsmodellen[36] erzeugen. Für Vergleichszwecke sollen drei Modelle implementiert werden. Diese sind das Fahrzeugfolgemodell (vergleiche Abschnitt 2.3.3.2), das Wiedemann-Modell (vergleiche Abschnitt 2.3.3.3) und ein erweitertes Wiedemann-Modell (vergleiche Abschnitt 2.3.3.4). Weiterhin soll es einfach sein, das Modul um beliebig viele mikroskopische Verkehrsmodelle zu erweitern. Das Verkehrssimulationsmodul muss netzwerkfähig sein, da es mit anderen Softwarekomponenten in einem Verteilten System interagieren muss (vergleiche Abschnitt 3.3). Eine weitere Anforderung ist, dass im simulierten Verkehrsstrom ein Fahrzeug frei steuerbar sein soll. Dieses Fahrzeug wird in einem externen Programm von einem

[35] Die Klassenmodell- und Sequenzdiagramme in der Masterarbeit wurden mit der Software MagicDraw erstellt.

[36] Diese Modelle werden in Abschnitt 2.3.3 eingehend betrachtet.

Menschen oder einer KI gesteuert. Die anderen Fahrzeuge im Verkehrsstrom müssen aber auf dieses reagieren. Trotzdem soll es auch möglich sein, einen rein simulierten Verkehrsstrom ohne das extern gesteuerte Fahrzeug zu erzeugen. Der simulierte Verkehrsstrom soll auf Grundlage der errechneten Daten in einem anderen Modul visualisiert werden. Dadurch, und wegen der Anforderung des extern gesteuerten Fahrzeuges, ist Zeit ein bedeutender Faktor.
Um eine für das menschliche Auge flüssige Visualisierung zu gewährleisten, sind die Fahrzeugpositionsdaten mindestens 24-mal in der Sekunde zu berechnen und zu übertragen. Um einen gewissen Puffer zu haben, soll von 30 Bildern pro Sekunde ausgegangen werden. Das bedeutet, dass ein Berechnungszyklus maximal rund 33 Millisekunden beanspruchen darf. Außerdem müssen die Daten des extern gesteuerten Fahrzeugs hinreichend schnell aktualisiert werden, weil erst dann die Berechnung der Fahrzeuge, die von diesem beeinflusst werden, stattfinden kann. Aus diesen beiden zeitkritischen Anforderungen ergibt sich, dass das Verkehrssimulationsmodul in Echtzeit unter weichen Echtzeitbedingungen[37] ablaufen muss, wobei 33 Millisekunden als Grenze für eine maximale Berechnungszyklusdauer angesehen werden. Es genügt, nur weiche Bedingungen zu fordern, da ein kurzzeitiges Überschreiten des geforderten Zeitkriteriums auf beispielsweise 66 Millisekunden in der Visualisierung nur zu einem kleinen, tolerierbaren Ruckeln führen würde.[38] Daraus ergibt sich, dass das Verkehrssimulationsmodul multithreading-fähig sein muss. Dies entspricht auch dem aktuellen Stand der Technik. Durch das Ausnutzen der gängigen aktuellen Mehrkernprozessoren kann die Berechnungszykluszeit verringert werden. Grundsätzlich liegt das Hauptaugenmerk bei der Modellierung des Verkehrssimulationsmoduls auf Performanz und Erweiterbarkeit.

3.3 Einordnung in das Verteilte System „RoadMaker“

Das Verkehrssimulationsmodul stellt einen Teil des Verteilten Systems „RoadMaker“[39] dar. Es befindet sich gegenwärtig noch im Entwicklungsstadium. Mit „RoadMaker“ soll eine möglichst wirklichkeitsnahe Fahrsimulation durchführbar sein. Dazu müssen die Straßen, das von einem Menschen gesteuerte Fahrzeug und die Verkehrssituation so realitätsnah wie möglich abgebildet, simuliert und fotorealistisch visualisiert werden.
Für die Realisierung dieses komplexen Systems wurde eine verteilte Struktur mit möglichst autonomen Modulen gewählt, weil so dieser sehr rechenintensive Gesamtprozess auf mehrere einzelne Rechner verteilt werden kann. Die Kommunikation zwischen den Modulen geschieht über ein Netzwerk. Die Simulation des vom Menschen gesteuerten Fahrzeugs wird mit der Software CarMaker (siehe Abschnitt 2.4.5.2) durchgeführt. Das Visualisierungsmodul wurde in einer weiteren parallel zu dieser Masterarbeit laufenden Masterarbeit (siehe [Kub10]) realisiert. Die Versorgung des Visualisierungsmoduls und des Verkehrssimulationsmoduls mit

[37] Für die Definition von Echtzeit und weichen Echtzeitbedingungen vergleiche Abschnitt 2.4.3.
[38] Vergleiche Abschnitt 4.8 für das tatsächliche Zeitverhalten des Verkehrssimulationsmoduls.
[39] „RoadMaker“ ist der vorläufige Arbeitsname.

den benötigten Informationen von CarMaker wird durch ein Kommunikationsmodul, welches im Institut für Verkehrssystemtechnik an der Westsächsischen Hochschule Zwickau entwickelt wurde, gewährleistet. Bei einem Austausch der Fahrzeugsimulationssoftware müsste so nur das Kommunikationsmodul überarbeitet werden. Vorübergehend findet noch eine direkte Kommunikation zwischen dem Visualisierungsmodul und dem Verkehrssimulationsmodul statt (siehe Abbildung 3.1). Ausgetauscht werden zwischen ihnen die Daten, die das Visualisierungsmodul für die Darstellung der Fahrzeuge benötigt. In der zukünftigen Entwicklung soll jedoch die gesamte intermodulare Kommunikation durch das Kommunikationsmodul gesteuert werden. Es ist auch vorgesehen, die in CarMaker vorhandenen Funktionalitäten für die Verkehrsabbildung zu nutzen, um die berechneten Daten des Verkehrssimulationsmoduls für die Steuerung der Verkehrsobjekte in CarMaker verwenden zu können.

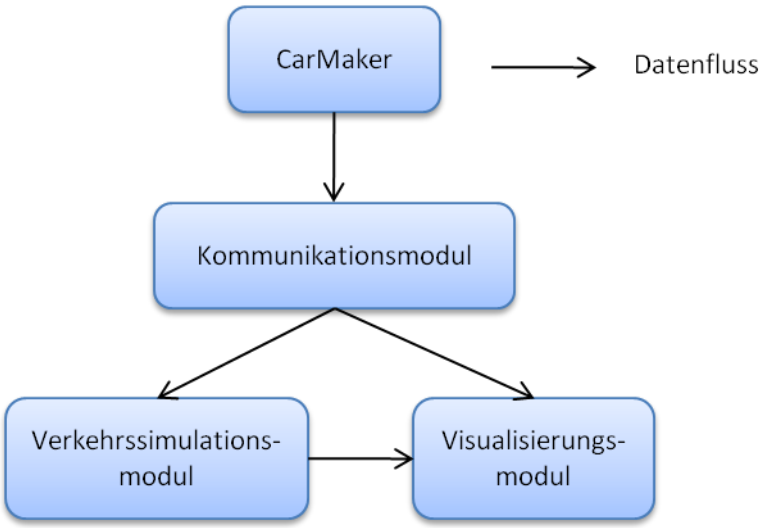

Abbildung 3.1: Stark vereinfachte Darstellung der gegenwärtigen Modulstruktur

Im weiteren Verlauf dieser Masterarbeit soll die intermodulare Kommunikation nur noch aus Sicht des Verkehrssimulationsmoduls betrachtet werden.[40]

3.4 Simulationsschleife

Bevor im folgenden Abschnitt die einzelnen Modellkomponenten genau betrachtet werden, erfolgt zunächst eine Erläuterung des groben Ablaufs der Simulationsschleife anhand der Abbildung 3.2, Seite 43. Ehe in die reguläre Simulationsschleife eingetreten werden kann, müssen alle Initialisierungen und sämtliche Instanziierungen abgeschlossen sein und alle benötigten Informationen zur Verfügung stehen.
Während eines Schleifendurchlaufs werden im Wesentlichen zwei Schritte ausgeführt. Als erstes erfolgt die Berechnung der FahrerFahrzeugdaten (1 bis 7)[41]. Im zweiten Schritt werden

[40] Ausführliche Erläuterungen dazu werden in den Abschnitten 3.5.6 und 4.4.5 gegeben.
[41] Die Zahlen in Klammern entsprechen den nummerierten Aufrufen in Abbildung 3.2, Seite 43.

dann die für die Visualisierung benötigten FahrerFahrzeugdaten an das Visualisierungsmodul gesendet (11, 12).

Nun erfolgen genauere Erläuterungen zur Berechnung der FahrerFahrzeugdaten. Zunächst müssen die benötigten Daten des externen Fahrzeuges vom Kommunikationsmodul erfragt werden (2). Dann sind die FahrerFahrzeuge, die den betrachteten Streckenabschnitt verlassen haben, aus dem Verkehrsstrom zu entfernen und die am Streckenbeginn eventuell einfahrenden FahrerFahrzeuge dem Verkehrsstrom hinzuzufügen (4). Danach kann die eigentliche Berechnung der FahrerFahrzeugdaten erfolgen (6, 7). Diese Berechnungen finden zu einem gewissen Maß parallelisiert statt.[42]

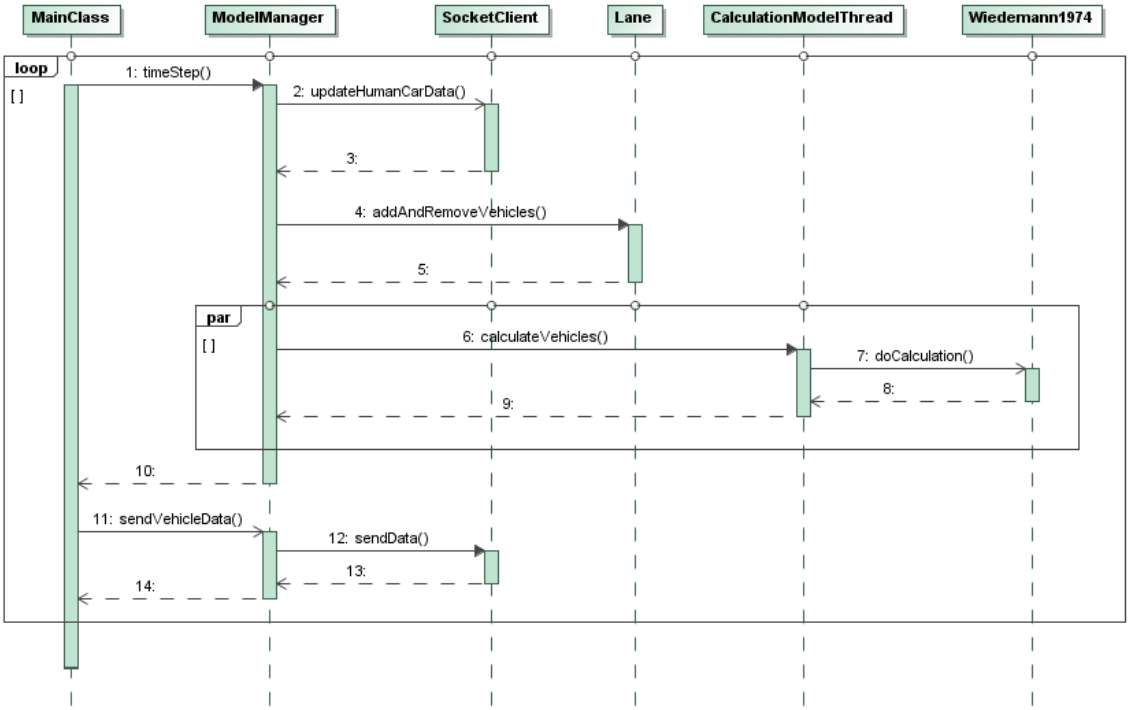

Abbildung 3.2: Vereinfachtes Sequenzdiagramm der Simulationsschleife

Die Simulation erfolgt in Zeitschritten mit nicht konstanter Zeitschrittdauer. Dabei wird für die Zeitdauer des aktuellen Simulationszeitschrittes die Zeitdauer des gerade abgelaufenen verwendet. Dies beruht auf der Annahme, dass die Zeit für die Berechnung von Zeitschritt zu Zeitschritt keinen großen Schwankungen unterliegt. Durch diese Schwankungen entstehen bei jedem Berechnungszeitschritt jedoch kleine Fehler. Ein extremes Beispiel: Der letzte Zeitschritt dauerte 5 Millisekunden, also wird im aktuellen Zeitschritt berechnet, wie weit sich ein FahrerFahrzeug während dieser Zeit bewegt. Angenommen, das FahrerFahrzeug hat eine Geschwindigkeit von 40 m/s und der aktuelle Zeitschritt eine Dauer von 20 Millisekunden, so wurde das FahrerFahrzeug 60 Zentimeter zu wenig bewegt. Für den darauf folgenden Zeitschritt wird jetzt eine Dauer von 20 Millisekunden angenommen. Tatsächlich dauert die

[42] Ausführlich wird die Parallelisierung durch die Abschnitte 3.5.5 und 4.4.4 beschrieben.

Berechnung aber nur 5 Millisekunden, wodurch nun das FahrerFahrzeug um 60 Zentimeter zu weit bewegt wurde. Damit wäre der Fehler vom letzten Zeitschritt wieder ausgeglichen.
Aus diesem Beispiel lassen sich zwei Dinge erkennen. Erstens, dass selbst bei sehr hohen Geschwindigkeiten und einer sehr großes Zeitdifferenz zwischen angenommener und tatsächlicher Zeitdauer noch relativ kleine Fehler entstehen. Zweitens, dass sich die auftretenden Fehler über die Zeit von selbst korrigieren, da die tatsächlich vergangene Zeitdauer mal größer und mal kleiner als die angenommene Zeitdauer ist. Die Aussage des „relativ kleinen Fehlers“ ergibt sich aus der Betrachtung des Anwendungskontexts. Für die simulierten FahrerFahrzeuge ist dieser Fehler ohne Bedeutung, da jedes den gleichen Fehler aufweist und ihre Positionen relativ zueinander nicht betroffen sind. Die Bedeutung des Fehlers bezieht sich auf den menschlichen Fahrer. Er sieht das vor ihm fahrende Fahrzeug um diesen Fehler versetzt zu nah oder zu weit weg. Dies liegt daran, dass das Fahrzeug des menschlichen Fahrers nicht unter der Kontrolle des Verkehrssimulationsmoduls steht, sondern unter der von CarMaker. Wenn also der menschliche Fahrer mit 40 m/s hinter einem Fahrzeug fährt, wird er einen Abstand von etwa 75 Metern zu diesem einhalten. Somit ist es für die Wahrnehmung des Fahrers unbedeutend, ob dieses Fahrzeug nun 60 Zentimeter zu nah oder zu weit entfernt angezeigt wird. Der menschliche Fahrer kann bei diesen kurzen Bildwiederholungsraten die konkreten, diskreten Positionsänderungen sowieso nicht wahrnehmen, sondern interpretiert sie als eine fließende Bewegung. Da sich die Fehler über die Zeit gegenseitig aufheben, kann also behauptet werden, dass sie für die Berechnung eines Zeitschrittes unter diesen Gesichtspunkten vernachlässigt werden können.[43]
Abschließend sollen die beiden Gründe genannt werden, weshalb keine konstante Zeitschrittdauer gewählt wurde:

1. Simulationszeit und Realzeit synchron zu halten ist bei diesem Verfahren kompliziert und aufwändig umzusetzen. Durch den erhöhten Rechenaufwand bliebe weniger Zeit für die eigentlichen Berechnungen.
2. Es müsste eine Synchronisierungen zwischen CarMaker und dem Verkehrssimulationsmodul durchgeführt werden, um Insynchronitäten zwischen beiden Simulationszeitverläufen zu korrigieren.

Dadurch wäre zwar sichergestellt, dass zwischen dem CarMaker-Fahrzeug und den simulierten Fahrzeugen nahezu keine Positionierungsfehler bestünden. Bei der Abwägung zwischen beiden Varianten ist der Autor jedoch zu dem Ergebnis gekommen, dass die hohe Genauigkeit bei konstanter Zeitschrittdauer den höheren Aufwand und die stärkere Kopplung der Module nicht aufwiegt.

[43] Siehe Abschnitt 4.8 für die Bestätigung dieser Aussage durch Messergebnisse.

3.5 Modellkomponenten

3.5.1 Modellstruktur

Der Modellentwurf gliedert sich in zwei Untermodelle auf. Im Kernmodell (*CoreModel*) sind die Modellkomponenten zusammengefasst, die den funktionellen Kern des Verkehrssimulationsmoduls darstellen. Die in den Abschnitten 3.5.2 bis 3.5.5 aufgeführten Teilmodelle bilden dieses *CoreModel*. Das Kommunikationsmodell (*CommunicationModel*) enthält die Funktionalität, die für den Datenaustausch mit dem Kommunikationsmodul und dem Visualisierungsmodul über ein Netzwerk benötigt wird. Zwischen beiden Untermodellen besteht nur eine relativ lose Kopplung (vergleiche Abschnitt 3.5.5). Das *CommunicationModel* wird in Abschnitt 3.5.6 näher erläutert werden.

3.5.2 FahrerFahrzeug

Die Erläuterung des FahrerFahrzeug-Teilmodells soll anhand von
Abbildung 3.3, Seite 46 erfolgen. Aus Gründen der Übersicht sind die Funktionen nicht mit angegeben. Sie sind dem ausführlichen Klassendiagramm in Anhang D entnehmbar.
Fahrer und Fahrzeug werden in den mikroskopischen Verkehrsmodellen als eine Entität gesehen. Diese Sichtweise soll sich in diesem Teilmodell wiederfinden. Der Fahrer und sein Fahrzeug werden immer durch ein Objekt repräsentiert.
Eine andere Modellierung wäre auch vorstellbar. Der Fahrer und das Fahrzeug könnten in jeweils einer eigenen Klasse modelliert werden, sodass jedem Fahrzeug ein Fahrer zugeordnet ist. Dadurch könnte ein Fahrzeug mit verschiedenen Fahrertypen ausprobiert werden. Dies ist jedoch nur dann sinnvoll, wenn sowohl Fahrer als auch Fahrzeuge mit unterschiedlichen Charakteristika vom verwendeten Verkehrsmodell berücksichtigt werden. Im Fahrzeugfolgemodell haben die Fahrer jedoch gar keine spezifischen Charakteristika, die ihr Fahrverhalten steuern (vergleiche Abschnitt 2.3.3.2). Beim Wiedemann-Modell können über die normalverteilten Zufallsgrößen verschiedene Fahrertypen erzeugt werden (vergleiche Abschnitt 2.3.3.3). Gerade beim Wiedemann-Modell stellt sich allerdings die Frage, wo eine Größe wie die maximale Beschleunigung anzusiedeln wäre, da sie sowohl vom Fahrzeugtyp als auch vom Fahrertyp abhängt. Grundsätzlich werden bei den mikroskopischen Verkehrsmodellen die Fahrzeuge nicht detailliert betrachtet. Wenn also ein Fahrzeug im Modell detaillierter abgebildet würde, müsste eine Erweiterung der Verkehrsmodelle erfolgen die dem Rechnung trägt. Das würde aber auch zu einem erhöhten Berechnungsaufwand führen. Zudem stellt sich die Frage, ob der erhöhte Realitätsgrad, der durch eine detailliertere Abbildung des Fahrzeuges erreicht würde, die Qualität des erzeugten Verkehrsstroms bemerkbar erhöht. Aus diesen Gründen, und weil die mikroskopischen Verkehrsmodelle Fahrer und Fahrzeug als eine Einheit

betrachten, hat sich der Autor für die Modellierung des Fahrers und des Fahrzeugs in einer Klasse entschieden.

Jedes mikroskopische Verkehrsmodell charakterisiert ein FahrerFahrzeug durch unterschiedliche Größen. Dennoch gibt es einige Größen wie die Fahrzeuglänge, die universelle Eigenschaften eines jeden FahrerFahrzeuges darstellen. Zudem soll das Modul einfach um beliebige Verkehrsmodelle erweiterbar sein. Die Klassen des FahrerFahrzeug-Teilmodells stellen weitgehend reine Datenobjekte dar, die nur den Zustand eines FahrerFahrzeuges repräsentieren.

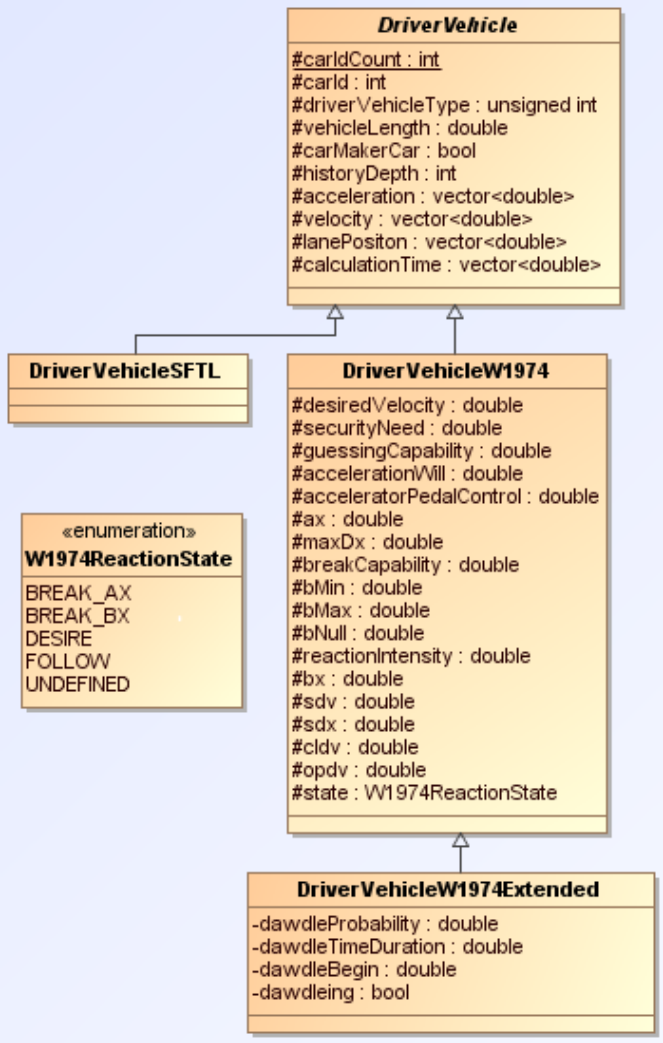

Abbildung 3.3: Vereinfachtes FahrerFahrzeug-Teilmodell

Die abstrakte Klasse *DriverVehicle* bildet eine einheitliche Schnittstelle zu dem FahrerFahrzeug-Teilmodell. Sie ist die Basisklasse von der die konkreten FahrerFahrzeug-Klassen, welche zu den jeweiligen korrespondierenden Verkehrsmodellen passen, direkt oder indirekt abgeleitet werden können. Auf diese Weise ist es möglich, beliebig viele unterschiedliche Repräsentationen von FahrerFahrzeugen zu implementieren, ohne dafür an anderen Stellen Veränderungen

vornehmen zu müssen. Die Klasse *DriverVehicle* umfasst die Variablen und Funktionen mit universellem Charakter.[44]

Tabelle 3.1: Variablen der *DriverVehicle* Klasse

Variablenname	Beschreibung
carIdCount	Anzahl der konkreten FahrerFarzeug-Instanzen
carId	Eineindeutiger Identifikator
vehicleLength	Die Länge des Fahrzeuges
driverVehicleType	Der FahrerFahrzeug-Typ; unter anderem dient es dem Visualisierungsmodul bei der Feststellung des Aussehens
carMakerCar	Hält fest, ob es sich um das von CarMaker gesteuerte FahrerFahrzeug handelt
acceleration	Die Beschleunigungshistorie des FahrerFahrzeugs
velocity	Die Geschwindigkeitshistorie des FahrerFahrzeugs
lanePosition	Die Streckenpositionshistorie des FahrerFahrzeugs
calculationTime	Die Berechnungszeitpunkthistorie
historyDepth	Anzahl an Historieneinträgen

Die Art, wie die Position eines FahrerFahrzeuges festgehalten wird, bedarf einer genaueren Betrachtung. Die Position des FahrerFahrzeugs wird relativ zum Beginn der befahrenen Strecke angegeben, wobei diese als eine Gerade betrachtet wird. Der Grund dafür ist, dass im Verkehrssimulationsmodul kein Streckennetzmodell vorhanden ist, da dessen Umsetzung im Rahmen der Masterarbeit nicht beabsichtigt war. Zudem werden in den für die Implementierung vorgesehenen Verkehrsmodellen die Auswirkungen von Kurven, Anstiegen und Gefällen einer Straße auf die Beschleunigung eines FahrerFahrzeuges nicht mit berücksichtigt. Damit die FahrerFahrzeuge in der Visualisierung dennoch auf der Straße fahren, muss die vom Verkehrssimulationsmodul berechnete Position entsprechend den geometrischen Eigenschaften der befahrenen Straße transformiert werden. Das geschieht im Visualisierungsmodul.
Beschleunigung, Geschwindigkeit und Position eines FahrerFahrzeuges werden nicht aktuell berechnet und dann im nächsten Zeitschritt wieder überschrieben. Ihr Verlauf (Historie) wird gespeichert, sodass für eine bestimmte Anzahl von Zeitschritten die Werte zurückverfolgt werden können. In der Member-Variablen *calculationTime* werden die jeweiligen Berechnungszeitpunkte festgehalten. Dadurch kann bei ausreichender Historientiefe bestimmt werden, welche Beschleunigung das FahrerFahrzeug vor 1.1 Sekunde hatte. Natürlich ist das aufgrund der Zeitschritte nicht exakt feststellbar. Zum Beispiel wurde die Beschleunigung nicht genau vor 1.1 Sekunde berechnet, sondern vor 1.096 Sekunden und vor 1.101 Sekunden. Diese Datenstruktur dient der Nachbildung der Reaktionszeit und wird in Abschnitt 4.4.3 erläutert.
Von *DriverVehicle* ist die Klasse *DriverVehicleSFTL* abgeleitet. Sie repräsentiert ein FahrerFahrzeug für das Fahrzeugfolgemodell (siehe Abschnitt 2.3.3.2). Es wird durch die Funktionalität

[44] Siehe Tabelle 3.1 für eine kurze Beschreibung der Variablen der Klasse.

der Elternklassen *DriverVehicle* nahezu vollständig abgedeckt. *DriverVehicleSFTL* implementiert lediglich die abstrakten Funktionen von *DriverVehicle* und einen Konstruktor für die korrekte Erzeugung einer *DriverVehicleSFTL* Instanz.
Die Klasse *DriverVehicleW1974* repräsentiert ein FahrerFahrzeug des Wiedemann-Modells (siehe Abschnitt 2.3.3.3). Die Variablen entsprechen den im Abschnitt 2.3.3.3 vorgestellten Bedeutungen und Bezeichnungen und sollen deshalb nicht weiter beschrieben werden. Dabei entsprechen die Member-Variablen *breakCapability* der Größe B_r und *reactionIntensity* der Größe R. Nur die Member-Variable *state* wurde zusätzlich eingeführt. In ihr wird festgehalten, welche Reaktion das FahrerFahrzeug gerade ausführt. *state* kann dabei die Werte einnehmen, die in der Enumeration *W1974ReactionState* aufgeführt sind. Die Bezeichnungen entsprechen den aus Abschnitt 2.3.3.3 bekannten. Der Wert UNDEFINED drückt aus, dass keine der vier Reaktionen durchgeführt wurde. *DriverVehicleW1974* implementiert einen Konstruktor für die korrekte Erzeugung der Instanzen der Klasse und die abstrakten Funktionen ihrer Elternklasse. Ansonsten werden lediglich Zugrifffunktionen auf die *DriverVehicleW1974* spezifischen Variablen definiert sowie eine Funktion zur Berechnung der zeitveränderlichen Variablen, wie die der Wahrnehmungsschwellen.
Von der Klasse *DriverVehicleW1974* ist die Klasse *DriverVehicleW1974Extended* abgeleitet. Sie repräsentiert ein FahrerFahrzeug für das erweiterte Wiedemann-Modell (siehe Abschnitt 2.3.3.4). Die Bedeutungen der Variablen gehen aus der Tabelle 3.2 hervor. Die Member-Variable *dawdleProbabillity* entspricht der in Abschnitt 2.3.3.4 beschriebenen Größe $z_{tröd}$ und *dawdleTimeDuration* der Größe $t_{tröd}$. Ansonsten überschreibt *DriverVehicleW1974Extended* nur den Konstruktor und die Funktion zur Berechnung der zeitveränderlichen Variablen der Elternklasse. Außerdem werden Zugrifffunktionen auf die *DriverVehicleW1974Extended* spezifischen Variablen definiert.

Tabelle 3.2: Variablen der *DriverVehicleW1974Extended* Klasse

Variablenname	Beschreibung
dawdleProbabillity	Die Wahrscheinlichkeit, mit der das FahrerFahrzeug zu trödeln beginnt
dawdleTimeDuration	Repräsentiert die Zeitdauer des Trödelvorgangs
dawdleBegin	Repräsentiert den Zeitpunkt des Trödelbeginns
dawdleing	Hält fest, ob das FahrerFahrzeug gegenwärtig trödelt

3.5.3 Fahrspur

Die Erläuterung des Fahrspur-Teilmodells soll anhand von Abbildung 3.4, Seite 49 erfolgen. Aus Gründen der Übersicht sind die Funktionen nicht mit aufgeführt. Sie können dem ausführlichen Klassendiagramm in Anhang D entnommen werden.

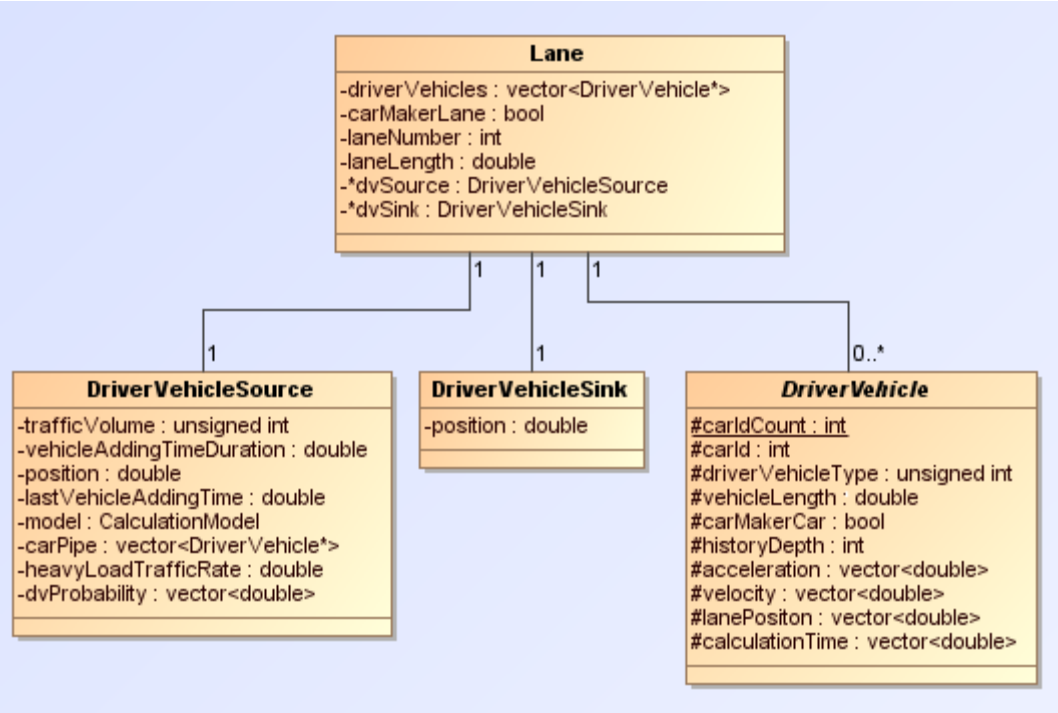

Abbildung 3.4: Vereinfachtes Fahrspur-Teilmodell

Eine Fahrspur wird durch eine Instanz der Klasse *Lane* repräsentiert und als Gerade mit einer bestimmten Länge angenommen.[45] Dabei können auf einer Fahrspur beliebig viele FahrerFahrzeuge fahren. Sie gelangen aus einer Quelle auf die Fahrspur und verschwinden an einer Senke wieder von ihr.

Tabelle 3.3: Variablen der *Lane* Klasse

Variablenname	Beschreibung
driverVehicles	Referenzen auf die FahrerFahrzeuge, die auf dieser Fahrspur fahren
carMakerLane	Gibt an, ob auf dieser Fahrspur das CarMaker-Fahrzeug fährt
laneNumber	Der eineindeutige Identifikator der Fahrspur
laneLength	Die Länge der Fahrspur
dvSource	Referenz auf die Quelle der FahrerFahrzeuge für diese Spur
dvSink	Referenz auf die Senke der FahrerFahrzeuge für diese Spur

Die Reihenfolge der FahrerFahrzeuge in der Member-Variablen *driverVehicles* entspricht dabei ihrer Reihenfolge auf der Fahrspur. Überholt ein FahrerFahrzeug ein anderes, müssen sie auch in *driverVehicles* die Positionen tauschen. Die Member-Variable *laneNumber* identifiziert eine Fahrspur auf folgende Weise (vergleiche Abbildung 3.5, Seite 50): Ein positiver Wert bedeutet, dass der Verkehr auf dieser Fahrspur in Fahrtrichtung des CarMaker-Fahrzeuges fließt, ein negativer Wert gibt an, dass er sich in Gegenfahrtrichtung des CarMaker-Fahrzeuges bewegt.

[45] Eine kurze Beschreibung der Variablen von *Lane* ist der Tabelle 3.3 zu entnehmen.

Die N Fahrspuren in Fahrtrichtung werden von links nach rechts mit 1 bis N, und die M Fahrspuren in Gegenfahrtrichtung rechts beginnend von -1 bis $-M$ bezeichnet.

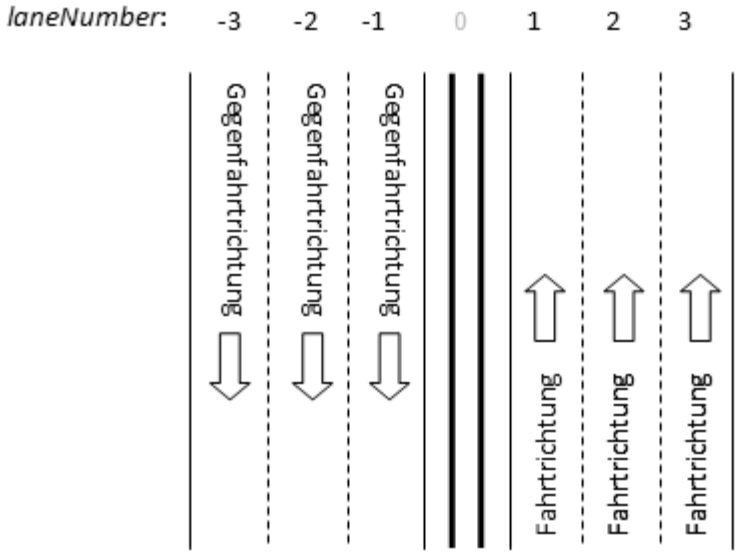

Abbildung 3.5: Beispiel für die Bedeutung des Wertes der Member-Variablen *laneNumber*

Die Klasse *DriverVehicleSource* repräsentiert eine auf der Fahrspur platzierte Quelle, die in Abhängigkeit von der eingestellten Verkehrsstärke FahrerFahrzeuge in bestimmten Zeitabständen auf die Fahrspur setzt.[46] Für den von der Quelle erzeugten FahrerFahrzeug-Strom können die Anteile des Schwerlastverkehrs und der jeweiligen FahrerFahrzeug-Typen angegeben werden. Auf einer Fahrspur lässt sich immer nur eine Quelle anbringen. Dabei ist sichergestellt, dass FahrerFahrzeug-Instanzen nur von den Klassen erzeugt werden, die mit dem eingestellten Verkehrsmodell korrespondieren.

Tabelle 3.4: Variablen der *DriverVehicleSource* Klasse

Variablenname	Beschreibung
trafficVolume	Die Verkehrsstärke, die die Quelle erzeugen soll
vehicleAddingTimeDuration	Die Zeitspanne, nach dem ein neues FahrerFahrzeug auf die Fahrspur gesetzt wird
position	Die Position der Quelle auf der Fahrspur
model	Information, welches Verkehrsmodell gerade benutzt wird
carPipe	Die FahrerFahrzeuge, die als nächstes auf die Fahrspur gesetzt werden
heavyLoadTrafficRate	Der Anteil des Schwerlastverkehrs am Gesamtverkehr
dvProbabillity	Die Anteile der jeweiligen FahrerFahrzeug-Typen am Gesamtverkehr

[46] Eine kurze Beschreibung der Variablen von *DriverVehicleSource* ist der Tabelle 3.4 zu entnehmen.

Die FahrerFahrzeuge fahren bis zum Erreichen einer Senke auf der Fahrspur. Die Klasse *DriverVehicleSink* repräsentiert diese Senke, welche sich beliebig auf der Fahrspur platzieren lässt. Haben die FahrerFahrzeuge die Senke erreicht, werden sie von der Fahrspur entfernt. Auf einer Fahrspur kann immer nur eine Senke platziert werden.

3.5.4 Verkehrsmodelle

Die Erläuterung des Verkehrsmodell-Teilmodells soll anhand von Abbildung 3.6 erfolgen.

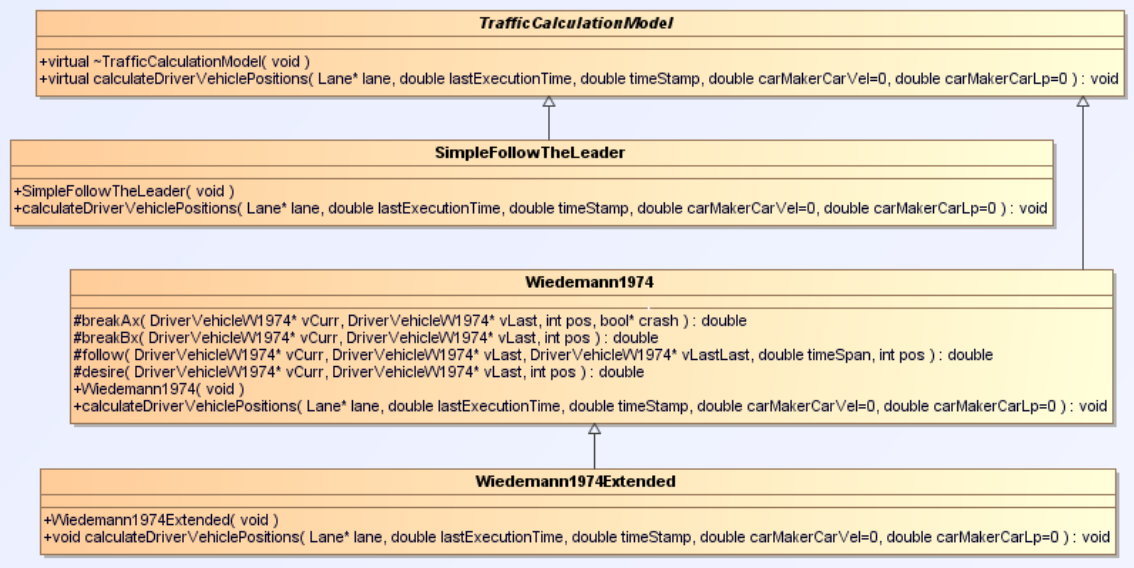

Abbildung 3.6: Verkehrsmodell-Teilmodell

Die abstrakte Klasse *TrafficCalculationModel* stellt eine einheitliche Schnittstelle zum Verkehrsmodell-Teilmodell dar. Von ihr werden die Klassen abgeleitet, welche die konkreten Implementierungen der Regelwerke und Berechnungsvorschriften der Verkehrsmodelle enthalten. Diese Klassen müssen dann die virtuelle Funktion *calculateDriverVehiclePositions(…)*[47] der Klasse *TrafficCalculationModel* implementieren, in der die FahrerFahrzeuge gemäß dem Modell bewegt werden. Dadurch können relativ einfach beliebig viele andere mikroskopische Verkehrsmodelle in das Verkehrssimulationsmodul integriert werden. Die Verkehrsmodelle bekommen in der Funktion *calculateDriverVehiclePositions(…)* eine Referenz auf eine Instanz von *Lane* übergeben, deren FahrerFahrzeuge entsprechend den Modellvorschriften bewegt werden. In dieser Funktion wird auch die Repräsentation des externen Fahrzeuges im Verkehrsstrom mit den Daten, die vom Kommunikationsmodul übertragen wurden, aktualisiert.

[47] „…“ bedeutet, dass die Funktion noch Aufrufparameter hat, die für die Erläuterungen aber mehrheitlich nicht relevant sind.

Die Klasse *SimpleFollowTheLeader* ist direkt von der Klasse *TrafficCalculationModel* abgeleitet und implementiert ein Fahrzeugfolgemodell-Verkehrsmodell. Die FahrerFahrzeuge werden gemäß der Gleichung (2.16) aus Abschnitt 2.3.3.2 bewegt.
Von der Klasse *TrafficCalculationModel* direkt abgeleitet ist auch die Klasse *Wiedemann1974*, welche das Wiedemann-Modell aus Abschnitt 2.3.3.3 implementiert. Die FahrerFahrzeuge werden nach dem dort erläuterten Regelwerk und dessen Berechnungsvorschriften bewegt.
Die Klasse *Wiedemann1974Extended* ist direkt von der Klasse *Wiedemann1974* abgeleitet. Sie erweitert das Wiedemann-Modell um das in Abschnitt 2.3.3.4 beschriebene Trödelverhalten der FahrerFahrzeuge und bewegt diese entsprechend des dort erläuterten Regelwerks.

3.5.5 Simulationssteuerung

Die Erläuterung des Simulationssteuerungs-Teilmodells soll anhand von Abbildung 3.7 erfolgen. Aus Gründen der Übersicht sind nicht alle Funktionen und Variablen mit aufgeführt. Sie können dem ausführlichen Klassendiagramm in Anhang D entnommen werden.

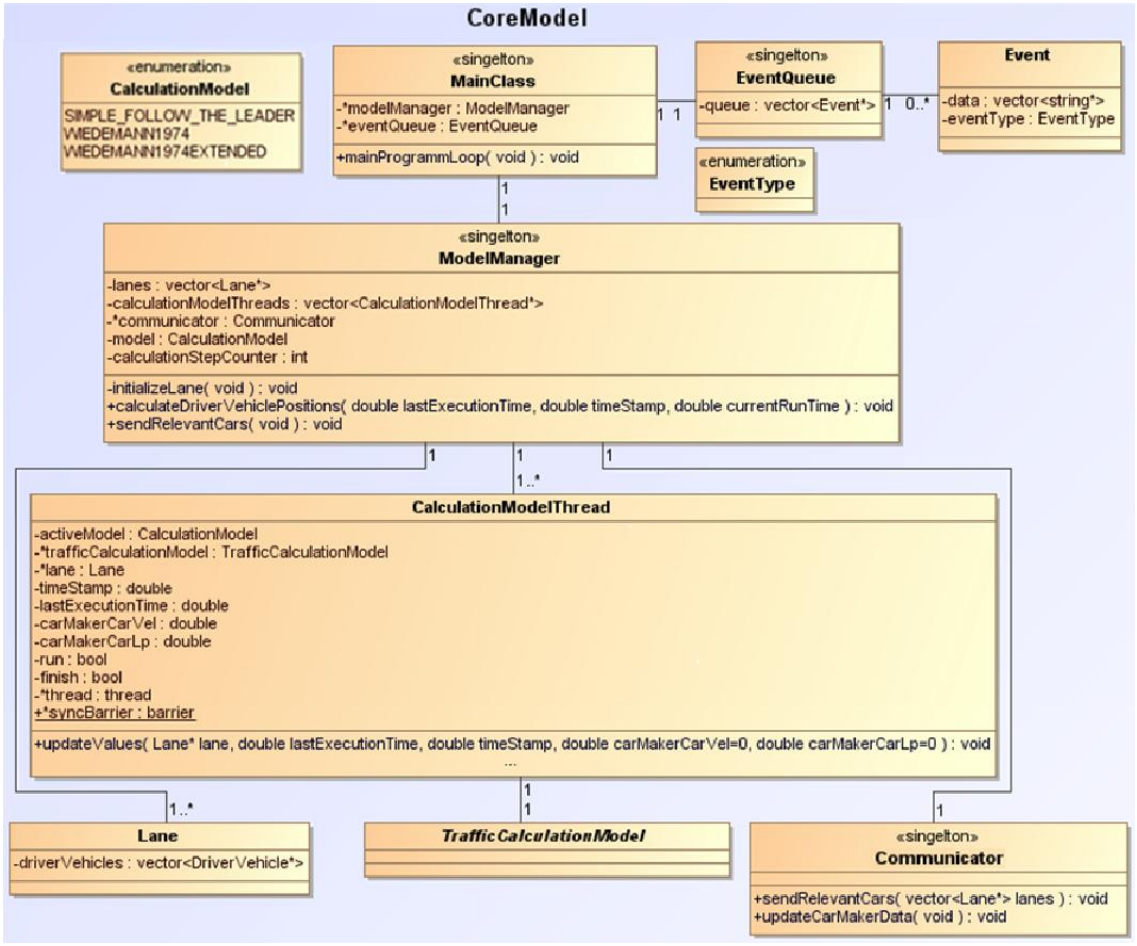

Abbildung 3.7: Vereinfachtes Simulationssteuerungs-Teilmodell[48]

[48] Die Klasse *Communicator* wurde aus Gründen der Darstellbarkeit als Teil des *CoreModel* abgebildet. Sie ist aber Teil des *CommunicationModel*; vergleiche Abschnitt 3.5.1 und 3.5.6.

Die im Verkehrssimulationsmodul implementierten Verkehrsmodelle sind in der Enumeration *CalculationModel* aufgelistet. Von der Klasse *MainClass* wird immer nur eine Instanz erzeugt. Sie hat als Variablen jeweils eine Referenz auf eine Instanz der Klasse *ModelManager* und *EventQueue*. *MainClass* hat die Aufgabe, die Simulationsschleife ablaufen zu lassen.
Die Steuerung der Simulation wird über die *ModelManager* Instanz durchgeführt. Über die Member-Variable vom Typ *EventQueue* können Ereignisse registriert und behandelt werden. Über sie können während der Simulation Eingriffe auf ihren Ablauf erfolgen. Gegenwärtig ist das allerdings noch nicht vorgesehen. Dieser rudimentäre Mechanismus soll die Erweiterbarkeit des Moduls erhöhen.
Auch von der Klasse *ModelManager* existiert immer nur eine Instanz.[49] Diese Klasse stellt die zentrale Steuerungsklasse dar. Sie regelt die komplette Ablaufsteuerung der Simulation. Über sie können die Fahrspuren mit einer initialen FahrerFahrzeug-Belegung ausgestattet, die Berechnungen der FahrerFahrzeuge ausgelöst und die Sendung der FahrerFahrzeug-Daten veranlasst werden. Dies ist die einzige Stelle, über die das *CoreModel* und das *CommunicationModel* miteinander verbunden sind.

Tabelle 3.5: Variablen der *ModelManager* Klasse

Variablenname	Beschreibung
lanes	Hält Referenzen auf alle Fahrspuren der Simulation
calculationModelThreads	Hält Referenzen auf alle Threads in der Simulation
communicator	Hält Referenz auf die Instanz der Klasse, die für die Kommunikation verantwortlich ist. Siehe Abschnitt 3.5.6
model	Information, welches Verkehrsmodell gerade benutzt wird
calculationStepCounter	Zählvariable für die Anzahl der durchgeführten Simulationsschritte

Durch den *ModelManager* erfolgen auch die Erzeugung und das Management der Threads. Für jede Instanz von *Lane* wird eine Instanz der Klasse *CalculationModelThread* erzeugt, in der die Berechnung der darauf befindlichen FahrerFahrzeuge zyklisch stattfindet.[50] Die Synchronisation der Threads geschieht durch ein Barrieren Konzept. Hat ein Thread einen Bearbeitungsdurchlauf beendet, stößt er gegen eine Barriere und wird aufgehalten. Erst nachdem auch der letzte Thread an der Barriere angekommen ist, können alle Threads ihre Arbeit fortsetzen.

[49] Eine Kurzbeschreibung der Variablen der *ModelManager* Klasse wird in Tabelle 3.5 gegeben.
[50] Eine Kurzbeschreibung der Variablen der *CalculationModelThread* Klasse wird in Tabelle 3.6, Seite 54 gegeben. Ihre genauen Bedeutungen werden im korrespondierenden Implementierungsabschnitt 4.4.4 erläutert.

Tabelle 3.6: Variablen der *CalculationModelThread* Klasse

Variablenname	Beschreibung
activeModel	Information, welches Verkehrsmodell gerade benutzt wird
trafficCalculationModel	Referenzen auf eine Instanz des verwendeten Verkehrsmodels
lane	Referenz auf die Instanz der Fahrspur, für die die Berechnung stattfinden soll
timeStamp	Der aktuelle Zeitpunkt
lastExecutionTime	Die Ausführungszeit des letzten Simulationsschrittes
carMakerCarVel	Geschwindigkeit des externen Fahrzeuges
carMakerCarPos	Die Position des externen Fahrzeuges
run	Gibt an, ob der Thread abläuft
finish	Gibt an, ob ein Abarbeitungsdurchlauf beendet ist
thread	Referenz auf die Instanz des eigentlichen Threads in dem die Berechnung stattfindet
syncBarrier	Referenz auf die Instanz der Barriere für die Thread Synchronisation

3.5.6 Kommunikationsmodell

Die Erläuterung des Kommunikationsmodells soll anhand von Abbildung 3.8 erfolgen.[51] Aus Gründen der Übersicht sind nicht alle Funktionen angegeben. Sie können dem ausführlichen Klassendiagramm in Anhang D entnommen werden.

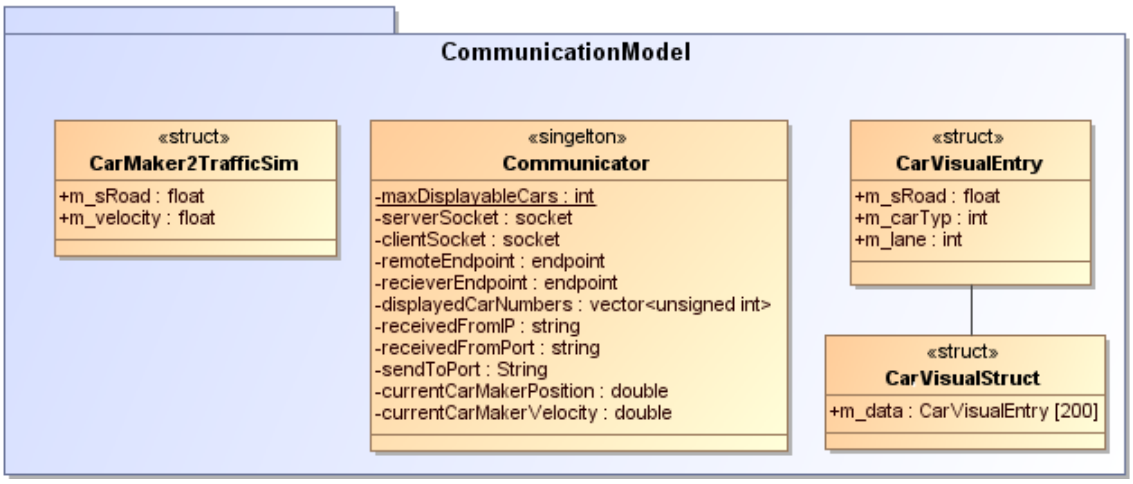

Abbildung 3.8: Vereinfachtes Kommunikationsmodell

[51] Eine Kurzbeschreibung der Variablen des *CommunicationModel* wird in Tabelle 3.7, Seite 56 gegeben. Ihre genauen Bedeutungen werden im korrespondierenden Implementierungsabschnitt 4.4.5 erläutert.

Von der Klasse *Communicator* wird nur eine einzige Instanz erzeugt. Über sie wird die gesamte Kommunikation des Verkehrssimulationsmoduls abgewickelt. *Communicator* verfügt über je einen Socket für die Kommunikation mit dem Visualisierungsmodul und dem Kommunikationsmodul. Der Datenaustausch mit dem Kommunikationsmodul findet über das Struct *CarMaker2TrafficSim* statt. Die Position des externen Fahrzeuges m*_sRoad* bezieht sich auch, wie in Abschnitt 3.5.2 erläutert, auf die Annahme der Strecke als Gerade. Das Visualisierungsmodul bekommt die Daten durch die Structs *CarVisualStruct* und *CarVisualEntry* übermittelt. Um das Datenvolumen zwischen Verkehrssimulationsmodul und Visualisierungsmodul zu verkleinern, werden nicht immer die Informationen sämtlicher FahrerFahrzeuge übertragen. Außerdem kann das Visualisierungsmodul nur eine begrenzte Anzahl an FahrerFahrzeugen flüssig darstellen. Die Auswahl der zu sendenden FahrerFahrzeug-Informationen findet vor dem Senden durch die *Communicator* Instanz statt. Diese Funktionalität ist im *CommunicationModel* angesiedelt, da diese keinerlei Relevanz für die eigentliche Kernfunktionalität der Verkehrsflusserzeugung hat, sondern aus Überlegungen zur Kommunikationsvereinfachung stammt. Die Datenstrukturen wurden mit den Verantwortlichen der jeweiligen Module abgesprochen.

Tabelle 3.7: Variablen der im *CommunicationModel* befindlichen Klassen und Strukturen

Variablenname	Beschreibung
CarMaker2TrafficSim	
m_sRoad	Die Position des externen FahrerFahrzeuges
m_velocity	Die Geschwindigkeit des externen FahrerFahrzeuges
CarVisualEntry	
m_sRoad	Die Position des simulierten FahrerFahrzeuges
m_carTyp	Der Typ des simulierten FahrerFahrzeuges
m_lane	Die Fahrspur, auf der das simulierte FahrerFahrzeuge fährt
CarVisualStruct	
m_data	Die für die Visualisierung bestimmten simulierten FahrerFahrzeuge
Communicator	
maxDisplayableCars	Anzahl an FahrerFahrzeugen, die vom Visualisierungsmodul flüssig dargestellt werden können
serverSocket	Das Socket für das Kommunikationsmodul
clientSocket	Das Socket für das Visualisierungsmodul
remoteEndpoint	Der Endpunkt für clientSocket
receiverEndpoint	Der Endpunkt für serverSocket
displayedCarNumbers	Die Anzahl der anzuzeigenden FahrerFahrzeuge pro Fahrspur vor und hinter dem externen Fahrzeug
receivedFromIP	Die IP des Kommunikationsmoduls
receivedFromPort	Der Port des Kommunikationsmoduls
sendToPort	Der Port des Visualisierungsmoduls
currentCarMakerPosition	Die aktuelle Position des externen FahrerFahrzeuges
currentCarMakerVelocity	Die aktuelle Geschwindigkeit des externen FahrerFahrzeuges

4 Implementierung des Verkehrssimulationsmoduls

4.1 Vorbemerkungen

Ausgehend vom Kapitel 3, in dem die Modellierung des Verkehrssimulationsmoduls beschrieben wurde, soll in diesem Kapitel umfassend auf seine Implementierung eingegangen werden. Die Entwicklung des Moduls erfolgte mit dem Microsoft Visual Studio 2008 unter Microsoft Windows 7 Professional 64-Bit. Als Programmiersprache wurde aufgrund der zeitkritischen Anforderungen C++ ausgewählt. Bei der Implementierung des Verkehrssimulationsmoduls wurden die Windows-API und die umfangreichen plattformübergreifenden boost-Bibliotheken in der Version 1.41 eingesetzt.[52] Das Verkehrssimulationsmodul ist unter Windows lauffähig. Es wurde aber darauf geachtet, so weit wie möglich plattformunabhängig zu bleiben, sodass bei einer Portierung des Moduls, zum Beispiel auf Linux, der Anpassungsaufwand relativ klein bleibt.

4.2 Modulstruktur

Das Verkehrssimulationsmodul besteht aus fünf Komponenten:

- In einer statischen Bibliothek ist die gesamte Funktionalität der Verkehrssimulation gekapselt. In ihr ist das Modell aus Kapitel 3 implementiert (siehe Abschnitt 4.4).
- Durch ein Konsolenprogramm wird die Simulation gestartet (siehe Abschnitt 4.5).
- Über eine Initialisierungsdatei kann die Simulation parametrisiert werden (siehe Abschnitt 4.6.1).
- Mit einer weiteren Initialisierungsdatei können verschiedene FahrerFahrzeug-Typen parametrisiert werden (siehe Abschnitt 4.6.2).

Diese vier Komponenten bilden den Kern des Verkehrssimulationsmoduls. Sie sind für eine korrekte Ausführung unabdingbar.

- Die fünfte Komponente ist eine GUI. Über sie werden die Initialisierungsdateien erstellt sowie das Konsolenprogramm zum Start der Simulation aufgerufen und terminiert (siehe Abschnitt 4.7).

Die GUI ist eine optionale Komponente, die für die Ausführung der Verkehrssimulation nicht benötigt wird. Sie dient ausschließlich dem Zweck der Erhöhung der Usability des Verkehrssimulationsmoduls.

Das Klassenmodell ist durch vier Namespaces strukturiert. Der erste Namespace `trafficSimulation` bildet das Toplevel-Element, dem die anderen drei untergeordnet sind und dem folglich auch alle Klassen angehören. Der zweite Namespace `core` umschließt alle

[52] Siehe http://www.boost.org/ für weitergehende Informationen. Über diese Website können sowohl die jeweils aktuelle als auch ältere Versionen runtergeladen werden.

Klassen, die die Kernfunktionalität des Moduls implementieren und ist identisch mit dem *CoreModel* aus Kapitel 3. Ein dritter Namespace `communication` enthält die Klassen, welche die Kommunikationsfunktionalität des Moduls implementieren. Er stimmt mit dem *CommunicationModel* aus Kapitel 3 überein. Der vierte `util` umfasst Werkzeugklassen, die bei der Implementierung der eigentlichen Funktionalität Hilfestellung leisten.

4.3 Zeitmessung

Die exakte Bestimmung der Zeit ist bei der Implementierung des Verkehrssimulationsmoduls von entscheidender Bedeutung. Deswegen wurden dafür die Funktionen `QueryPerformanceCounter(…)` und `QueryPerformanceFrequency(…)` verwendet. Die erste Funktion gibt den Wert des Performance Counters zurück und die zweite Funktion dessen Frequenz. Er ist ein Hardwarebaustein heutiger Mikroprozessoren und arbeitet im Megahertz-bereich.[53] Seine Frequenz kann sich aber auf verschiedenen Computern unterscheiden. Der Wert des Performance Counters ist eine Anzahl von Takten seit Einschalten des Computers. Durch die Bildung des Quotienten aus der Anzahl an Takten und der Frequenz kann die Zeit, die seit dem Einschalten des Computers vergangen ist, in Sekunden bestimmt werden. Da der Performance Counter im Megahertz-Bereich getaktet ist, wird so eine Zeitmessung im Nanosekundenbereich möglich. Das ist im Moment die präziseste Methode zur Zeitmessung.

4.4 Modellkomponenten

4.4.1 FahrerFahrzeug

Die Zuweisung der `carId` eines DVs[54] erfolgt über den Aufruf der statischen Funktion `generateCarId()` der abstrakten Klasse `DriverVehicle`, die den um eins inkrementierten Wert der statischen Variablen `carIdCount` zurückgibt. Die Beschleunigung eines DVs wird in der Member-Variablen `acceleration` in m/s², die Geschwindigkeit in `velocity` in m/s, die Position in `position` in Meter und die Berechnungszeit in `calculationTime` in Sekunden gespeichert. Diese Variablen sind vom Typ `std::vector<double>`[55] und haben die Länge `historyDepth`. Diese vector-Variablen speichern die berechneten Werte der mindestens letzten zwei Sekunden. Bearbeitet werden können diese vector-Variablen über zwei Template Funktionen, die sicherstellen, dass die Anzahl der Einträge mindestens die letzten zwei Sekunden umfassen.

[53] Gegenwärtig um die 3 Megahertz.
[54] Vereinfachend soll im Folgenden bei einem „DriverVehicle“ bzw. DV von der Instanz einer von der Klasse `DriverVehicle` direkt oder indirekt abgeleiteten Klasse gesprochen werden.
[55] Im Folgenden vereinfachend als vector bezeichnet.

Bei der Konstruktion eines DVs wird zunächst ein Speicherbereich von 100 Einträgen pro vector allokiert. Es wurde auch eine Variante mit konstanter vector-Länge untersucht. Es zeigte sich aber, dass der Ansatz mit dynamischer vector-Länge trotz der zur Laufzeit durchgeführten Speicherallokation schneller ist. Auf die vector-Variablen kann über zwei Funktionen zugegriffen werden. Die erste Funktion ist nach dem Schema `getCurrent<VariablenName>()` benannt. Sie gibt den ersten Eintrag des vectors zurück. Die zweite Funktion ist nach dem Schema `get<VariablenName>Before(int pos)` benannt und gibt den Eintrag des vectors an der Stelle `pos` zurück. Über die Funktion `getPositionByTimeDifference(double timeDifference, double currentTime)` der Klasse `DriverVehicle` kann der vector-Eintrag ermittelt werden, der ausgehend vom Zeitpunkt `currentTime` vor `timeDifference` Sekunden berechnet wurde. Desweiteren implementiert die Klasse `DriverVehicle` Zugriffsfunktionen für ihre Variablen.

```
void setAcceleration(double acc);
void setCurrentTime(double t);
void updateVelocity(double timeSpan);
void updateLanePosition(double timeSpan);
```

Listing 4.1: Deklaration der Aktualisierungsfunktionen von `DriverVehicle`

Die Aktualisierung der vector-Variablen erfolgt über die Funktionen aus Listing 4.1. Es können dabei nur die Beschleunigung und der Berechnungszeitpunkt direkt gesetzt werden. Die Geschwindigkeit und die Position des DVs werden bezogen auf die aktuelle Zeitschrittdauer `timeSpan` berechnet.[56] Bei der Aktualisierung der Geschwindigkeit wurde sichergestellt, dass diese nicht negativ werden kann. Über die Funktionen `setCarMakerCarVelocity(…)` und `setCarMakerCarLanePosition(…)` können Position und Geschwindigkeit des extern gesteuerten CarMaker-DVs direkt gesetzt werden. Ferner umfasst die Klasse `DriverVehicle` einige rein virtuelle Funktionen. Ihre Implementierungen in den Kindklassen sollen verschiedene Beschreibungen eines DVs in Form eines `std::string` zurückgeben.

```
DriverVehicleSFTL(unsigned int vt, double acc, double lp, bool carMakerCar);
```

Listing 4.2: Deklaration des Konstruktors der Klasse `DriverVehicleSFTL`

Die Klasse `DriverVehicleSFTL` implementiert ein FahrerFahrzeug des Fahrzeugfolgemodells. Die Deklaration ihres Konstruktors kann aus Listing 4.2 entnommen werden. Im Konstruktor werden die `carId` gesetzt, `vt` der `driverVehicleType` Member-Variablen sowie `carMakerCar` der `carMakerCar` Member-Variablen zugewiesen. Der FahrerFahrzeug-Typ gibt an, ob es sich um einen LKW oder PKW handelt und bestimmt die Länge des FahrerFahrzeuges sowie dessen Initialgeschwindigkeit. Diese Werte werden aus der Initialisierungsdatei für die FahrerFahrzeug-Typen bezogen (siehe dazu die Abschnitte 4.4.6 und 4.6.2). Die Geschwindigkeit wird aus der Initialisierungsdatei für diesen FahrerFahrzeug-Typ angegebenem Erwartungswert μ und der Standardabweichung σ über einen normalverteilten Zufallszahlengenerator bestimmt.

[56] Dies geschieht nach den Gleichungen (2.4) und (2.3) aus Abschnitt 2.2.4.

Der ermittelte Wert befindet sich in einer 3σ-Umgebung von μ. Die vector-Variable `velocity` wird mit diesem ermittelten Wert, `acceleration` mit `acc`, `lanePosition` mit `lp` und `calculationTime` mit der Konstruktionszeit initial gefüllt. Desweitern implementiert die Klasse `DriverVehicleSFTL` nur die virtuellen Funktionen aus der Elternklasse.

Die Klasse `DriverVehicleW1974` implementiert ein FahrerFahrzeug des Wiedemann-Modells (siehe Abschnitt 2.3.3.3). Die Parameterliste des Konstruktors entspricht bis auf `acc` der aus Listing 4.2, Seite 59 und ist, soweit es die gemeinsamen Variablen betrifft, ähnlich implementiert. Nur `acceleration` wird abweichend mit `0` initial gefüllt. Die Wiedmann-Modell spezifischen normalverteilten Zufallsvariablen wie `securityNeed` bekommen einen zufälligen $(0.5,0.15)$-normalverteilt erzeugten Wert zugewiesen. Nur die Wunschgeschwindigkeit wird aus den Werten für μ und σ der Initialisierungsdateien für die FahrerFahrzeug-Typen entnommen. Die Wahl dieser Werte, die die Wunschgeschwindigkeitsverteilung bestimmen, ist schwierig. Besonders bei Landstraßen kann aufgrund der verschiedenen geometrischen Verläufe nur sehr schwer eine allgemein verbindliche Wunschgeschwindigkeitsverteilung angegeben werden. Am besten ist es, sie aus empirischen Untersuchungen zu gewinnen. Die Member-Variablen `breakCapability` und `maxDx` werden gemäß Definition initialisiert. Der Member-Variablen `state` wird `UNDEFINED` zugewiesen und den restlichen `0`.

Die Berechnungen der Wahrnehmungsschwellen und Beschleunigungsgrenzen erfolgen in der Funktion `updateWiedemannThresholdVars(…)` gemäß dem Wiedemann-Modell. Für die Berechnung der Member-Variablen `ax` wurde die ursprüngliche Berechnungsgleichung (2.28) so abgewandelt, dass noch eine Addition von 2 zu dem berechneten Wert erfolgt. Das geschieht, da sich ohne diese Addition bei einem Stau ein leichtes Ineinanderrollen der DVs zeigt. Diese Abwandlung hat sonst keinen bemerkenswerten Einfluss, allenfalls führt sie zu einem geringfügig sichereren Fahrverhalten, weil entsprechend größere Sicherheitsabstände eingehalten werden.

Ein Problem stellt die Implementierung des Erhöhungsverhaltens der Member-Variablen `reactionIntensity` dar, welche der Größe R des Wiedemann-Modells entspricht. Ursprünglich wird die Größe R während eines Zeitschritts von einer Sekunde um 1 inkrementiert. Deswegen erfolgt die Erhöhung der Member-Variablen `reactionIntensity` in einem Zeitschritt anteilsmäßig um $1 \cdot Zeitschrittdauer$. Da `reactionIntensity` aber immer ab 0 hochgezählt wird und sie bei den Modellberechnungen im Nenner verwendet wird, käme es während des Hochzählens von 0 auf 1 zu extremen Verfälschungen. Aus diesem Grund wird `reactionIntensity` direkt auf 1 gesetzt, wenn der aktuelle Wert 0 ist. Anschließend erfolgt die zeitanteilsmäßige Erhöhung. Im Weiteren implementiert die Klasse `DriverVehicleW1974` nur Zugriffsfunktionen für die `DriverVehicleW1974` spezifischen Variablen und die virtuellen Funktionen aus der Elternklasse.

Die Klasse `DriverVehicleW1974Extended` implementiert ein FahrerFahrzeug des erweiterten Wiedemann-Modells (siehe Abschnitt 2.3.3.4). Die Parameterliste des Konstruktors entspricht bis auf `acc` der aus Listing 4.2, Seite 59. Ansonsten ist der Konstruktor wie der der Klasse `DriverVehicleW1974` implementiert. Die `DriverVehicleW1974Extended` spezifischen

Variablen werden folgendermaßen initialisiert: `dawdleBegin` und `dawdleing` werden grundsätzlich auf `0` beziehungsweise `false` gesetzt. Die Member-Variable `dawdleProbability` wird bei LKW-FahrerFahrzeug-Typen auf `1` und `dawdleTimeDuration` auf `0` gesetzt, da LKW nicht trödeln. Bei PKW-FahrerFahrzeug-Typen erfolgt die Bestimmung der Werte für die Member-Variablen `dawdleProbability` und `dawdleTimeDuration` über einen gleichverteilten Zufallszahlengenerator. Der Wert für `dawdleProbability` bewegt sich dabei zwischen `0.95` und `0.99` und für `dawdleTimeDuration` zwischen `25` und `40` Sekunden. Auch die Implementierung der Funktion `updateWiedemannThresholdVars(…)` stimmt mit der der Elternklasse überein, wobei aber die Berechnungen der Member-Variablen `bMax` und `bMin` nach dem im Abschnitt 2.3.3.4 angegebenen Gleichungen erfolgen. Ansonsten implementiert die Klasse `DriverVehicleW1974Extended` nur Zugriffsfunktionen für die spezifischen Variablen dieser Klasse.

4.4.2 Fahrspur

Die Klasse `Lane` hat zwei Konstruktoren. Die Deklaration des ersten ist in Listing 4.3 aufgeführt. Der zweite Konstruktor ist bis auf den ersten Parameter gleich deklariert. Beide Konstruktoren weisen die Parameter den gleichlautenden Member-Variablen der Klasse `Lane` zu. Es werden eine neue Instanz von `DriverVehicleSink` an der Stelle `sinkPosition` erzeugt und ein Zeiger darauf der Member-Variablen `dvSink` zugewiesen. In `dvSource` wird ein Zeiger auf eine neu erzeugte Instanz von `DriverVehicleSource` gespeichert. Diese Quelle an der Position `sourcePosition` erzeugt DVs, die zum Verkehrsmodell `model` passen mit einer Verkehrsstärke von `trafficVolume` und einem SV-Anteil von `heavyLoadTrafficRate`. Desweiteren sind in der Klasse `Lane` verschiedene Funktionen, die den Zugriff auf die DVs in `driverVehicels` erleichtern sowie Zugriffsfunktionen für die anderen Variablen außer für `dvSource` und `dvSink` implementiert. Das Hinzufügen und Löschen von DVs aus `driverVehicles` obliegt allein der Quelle und der Senke der `Lane` Instanz.

```
Lane(std::vector<DriverVehicle*> driverVehicles, int laneNumber,
     double laneLength, unsigned int trafficVolume,
     double sourcePosition, double sinkPosition,
     CalculationModel model, double heavyLoadTrafficRate,
     bool carMakerLane = false
   );
```

Listing 4.3: Deklaration des Konstruktors der Klasse `Lane`

Über die erste Funktion aus Listing 4.4, Seite 62 kann die Quelle der Fahrspur angewiesen werden, sie mit einer initialen DV-Belegung zu füllen. Die zweite Funktion weist die Quelle an, anhand von `currentRunTime` zu überprüfen, ob ein DV am Beginn der Fahrspur hinzugefügt werden soll. Mit der dritten Funktion wird die Senke der Fahrspur angewiesen zu überprüfen, ob ein DV das Ende der Fahrspur erreicht hat und es somit aus der `driverVehicles` Variablen entfernt werden muss.

```
void initializeDriverVehicleOnLane(void);
void addDriverVehicle(double currentRunTime);
void removeDriverVehicles(void);
```

Listing 4.4: Deklaration der Verkehrsstrom-Management Funktionen der Klasse `Lane`

Die Implementierung der Klasse `DriverVehicleSink` ist nicht besonders Umfangreich (siehe Listing 4.5). Bei der Konstruktion einer Instanz wird lediglich der Wert des Parameters des Konstruktors der Member-Variablen `position` zugewiesen. Beim Aufruf der Funktion `removeDriverVehicles(…)` werden die DVs der übergebenen `Lane` Instanz danach überprüft, ob ihre jeweilige Position auf der Fahrspur größer der Member-Variablen `position` ist. In diesem Fall wird dieses DV aus `lane` entfernt und anschließend gelöscht.

```
class DriverVehicleSink
{
private: double position;
public:
   DriverVehicleSink(double position);
   void removeDriverVehicles(Lane* lane);
};
```

Listing 4.5: Deklaration der Klasse `DriverVehicleSink`

Im Konstruktor der Klasse `DriverVehicleSource` (siehe Listing 4.6, Seite 63) werden zunächst alle Werte der Parameter den jeweils gleichlautenden Member-Variablen zugewiesen. Der Wert von `vehicleAddingTimeDuration` errechnet sich aus dem Quotienten von `3600` und `trafficVolume` und enthält so die Zeitdauer in Sekunden, nach der ein DV auf die dazugehörige Fahrspur hinzugefügt werden muss. `lastVehicleAddingTime` wird mit `0` initialisiert und der Speicherbereich für `dvProbabilitys` allokiert. Anschließend wird die Funktion `setAndAdjustDvProb()` aufgerufen und in `carPipe` 1000 DVs gespeichert, die mit `createDriverVehicle(…)` an der Stelle `position` erzeugt wurden. Durch die Erzeugung von 1000 DVs während der Initialisierungsphase der Simulation können je nach Verkehrsstärke dutzende Minuten simuliert werden, ohne dass dabei Zeit für die Erzeugung von DVs verbraucht wird.

Die Funktion `setAndAdjustDvProb()` setzt die Wahrscheinlichkeiten in `dvProbabilitys`, mit der ein DV eines bestimmten Typs erzeugt wird und justiert diese gegebenenfalls. Durch ungeschickte Parameterwertsetzung in den Initialisierungsdateien können der SV-Anteil der Quelle mit der Summe der Anteile von LKW DVs nicht übereinstimmen. Daraufhin werden die Wahrscheinlichkeiten in `dvProbabilitys`, mit der ein DV eines bestimmten Typs erzeugt wird, angepasst. Die Wahrscheinlichkeiten der LKW-Typen werden dann auf $\frac{SV - Anteil}{Anzahl\ LKW - Typen}$ gesetzt. Falls dadurch nun die Summe der Wahrscheinlichkeiten kleiner 100 ist, wird die Differenz dem ersten PKW-Typ hinzuaddiert. Ist die Summe aber größer als 100, wird diese Differenz beim ersten PKW-Typ abgezogen. Sinkt dessen Wahrscheinlichkeit daraufhin auf 0 und die Summe ist weiterhin größer 100, so wird mit der Subtraktion beim nächsten PKW-Typ fortgefahren und das so lange, bis die Summe der Wahrscheinlichkeiten 100 beträgt.

```
class DriverVehicleSource
{
private:
   DriverVehicle* createDriverVehicle(double lanePosition,
                                      bool carMakerCar);
   void setAndAdjustDvProb(void);
public:
   DriverVehicleSource(unsigned int trafficVolume, double position,
                       CalculationModel model, double heavyLoadTrafficRate);
   void addDriverVehicle(Lane* lane, double currentRunTime);
   void initialOccupancy(Lane* lane, double laneLength);
};
```

Listing 4.6: Deklaration des Konstruktors und der Funktionen der Klasse `DriverVehicleSource`

Die Funktion `createDriverVehicle(…)` erzeugt ein DV, das mit dem eingesetzten Verkehrsmodell korrespondiert und gibt einen Zeiger darauf zurück. Das DV wird an der Position `lanePosition` erzeugt. Der FahrerFahrzeug-Typ wird aus den Wahrscheinlichkeiten in `dvProbabilitys` bestimmt. Sollten DVs vom Typ `DriverVehicleSFTL` erzeugt werden, wird dem Konstruktor für `acc` ein zufälliger, gleichverteilt bestimmter Wert zwischen `-1` und `1` übergeben. Hat der Parameter `carMakerCar` der Funktion `createDriverVehicle(…)` den Wert `true` und impliziert damit, dass das zu erzeugende DV das extern gesteuerte CM-DV repräsentieren soll, bekommt es als FahrerFahrzeug-Typ den Wert $Anzahl\ FahrerFahrzeug-Typen+1$ übergeben. Die restlichen Parameter werden `0` beziehungsweise `true` gesetzt.

Die Funktion `initialOccupancy(…)` belegt die Fahrspur beginnend bei `laneLength` bis `0` mit DVs. Die Platzierung erfolgt in einem Abstand, der zufällig, gleichverteilt zwischen `100` und `180` Metern liegt. Falls es sich bei der zur Quelle gehörenden Fahrspur um die handelt, auf der das CM-DV fährt, wird es hierbei erzeugt.

Die Funktion `addDriverVehicle(…)` fügt der Fahrspur ein DV aus `carPipe` an der Stelle `position` hinzu, wenn die Zeitspanne `vehicleAddingTimeDuration` vergangen ist und zum direkt vorausfahrenden DV der Abstand mindestens 15 Meter beträgt. Sollte die Anzahl an DVs in `carPipe` auf unter 10 fallen, werden 20 neue DVs erzeugt und `carPipe` hinzugefügt. Aufgrund des Mindestabstandes von 15 Metern bevor ein neues DV hinzugefügt wird, können sehr hohe Verkerhsstärken (> 3000 Fz/h) nicht erzeugt werden.

4.4.3 Verkehrsmodelle

In jedem der implementierten Verkehrsmodelle bestimmen die Werte zum Zeitpunkt t erst nach einer Reaktionszeit T_r zum Zeitpunkt $t+T_r$ das Verhalten eines FahrerFahrzeuges. Umgekehrt lässt sich jedoch auch sagen, dass das Verhalten eines FahrerFahrzeuges zum aktuellen Zeitpunkt t von den Werten des Zeitpunktes $t-T_r$ bestimmt wird. Diese Methode wurde implementiert. Die Nachbildung der Reaktionszeit erfolgt in allen Modellen nach folgendem Schema:

1. Durch Aufruf der Funktion `getPositionByTimeDifference(double timeDifference, double currentTime)` mit der Reaktionszeit T_r für `timeDifference` und t für `currentTime` wird der Index der vector-Variablen der DVs bestimmt, an der die vor T_r berechneten Werte stehen.
2. Durch Aufruf der Funktion `get<VariablenName>Before(int pos)` mit dem aus 1. ermittelten Index für `pos` können nun die jeweiligen Werte für Beschleunigung, Geschwindigkeit und Position eines DVs vor T_r Sekunden ermittelt werden.

Alle Verkehrsmodelle sind in Klassen implementiert, die direkt oder indirekt von `TrafficCalculationModel` abgeleitet sind. Diese deklariert nur die rein virtuelle Funktion `calculateDriverVehiclePositions(…)` (siehe Listing 4.7).

```
void calculateDriverVehiclePositions(Lane* lane, double lastExecutionTime,
                                     double timeStamp, double carMakerCarVel = 0,
                                     double carMakerCarLp = 0);
```

Listing 4.7: Deklaration der Verkehrsberechnungsfunktion in den Verkehrsmodell-Klassen

Das Fahrzeugfolgemodell ist in der Klasse `SimpleFollowTheLeader` in der Funktion aus Listing 4.7 programmiert. Vor Beginn der Berechnung wird geprüft, ob die DVs in `lane` vom zu diesem Verkehrsmodell passenden Typ `DriverVehicleSFTL` sind. Ist das nicht so, bricht die gesamte Simulation ab. Andernfalls erfolgt in einer Schleife die eigentliche Berechnung für jedes DV aus `lane`. Als erstes wird überprüft, ob es sich um das CM-DV handelt. In diesem Fall wird als aktueller Beschleunigungswert die Differenz aus `carMakerCarVel` und dem Geschwindigkeitswert aus dem letzten Simulationsschritt zugewiesen. Als aktuelle Geschwindigkeit und Position werden jeweils die Parameterwerte `carMakerCarVel` und `carMakerCarLp` sowie `timeStamp` als Berechnungszeitpunkt gesetzt.

Bei einem simulierten DV wird dessen Beschleunigung gemäß Gleichung (2.16) aus Abschnitt 2.3.3.2 berechnet. Bei Tests zeigte sich, dass diese Berechnung der DV Beschleunigungen allein nach Gleichung (2.16), relativ schnell zu einem vollkommen unrealistischen Verkehrsstrom führte. Aus diesem Grund wurde eine obere Grenze für die Geschwindigkeit festgelegt. Fährt ein DV schneller als 70 m/s, so wird dessen Beschleunigung auf $-1 \cdot$ `lastExecutionTime` gesetzt.

Das erste DV von `lane` müsste gemäß Modell immer mit 0 beschleunigen. Um jedoch die modellbedingte Anpassung der Geschwindigkeiten der DVs an das erste DV etwas aufzulockern, wird dessen Beschleunigung auf einen zufälligen, gleichverteilten Wert zwischen `-1` und `1` gesetzt. Erfolgt keine Ausnahmebehandlung, errechnet sich die Beschleunigung der DVs gemäß Gleichung (2.16). Dabei mussten allerdings die Reaktionszeit und der Sensitivitätsfaktor κ angepasst werden. In Verkehrsströmen ohne das CM-DV zeigte sich, dass eine Reaktionszeit von einer Sekunde und ein Sensitivitätsfaktor zwischen `1` und `0.2` relativ gute Ergebnisse liefern. Auf der Fahrspur mit dem CM-DV wurde für die Reaktionszeit `0.2` Sekunden und als Sensitivitätsfaktor `1` gewählt. So kommt es bei einem Abbremsen des CM-DVs nicht mehr so häufig zu einem Unfall mit den nachfolgenden DVs. Es wurde auch getestet, den Sensitivitätsfaktor so hoch einzustellen, dass es zu gar keinen Unfällen zwischen dem CM-DV und den

nachfolgenden DVs bei einem plötzlichen Abbremsen kommt. Allerdings führte dies auch dazu, dass DVs unvermittelt anhielten. Ein Sensitivitätsfaktor von 1 stellt einen guten Kompromiss zwischen Unfallvermeidung und fließendem Verkehr dar. Um die Wahrscheinlichkeit für einen Zusammenstoß mit dem CM-DV noch weiter zu verringern, wird die Beschleunigung eines DVs auf $-15 \cdot$ `lastExecutionTime` gesetzt sobald es sich auf weniger als 30 Meter seinem Vorderfahrzeug genähert hat. Dies geschieht allerdings nur auf der Fahrspur des CM-DV.
Am Ende eines jeden Schleifendurchlaufs werden die ermittelte Beschleunigung und der Berechnungszeitpunkt im DV gesetzt sowie die Funktionen zur Aktualisierung der Geschwindigkeit und der Position aufgerufen. Nachdem die Daten aller DVs aktualisiert sind, werden sie mittels des QuickSort-Algorithmus' der Position nach absteigend in der Member-Variable `driverVehicles` von `lane` sortiert. Dies muss erfolgen, da das CM-DV ein anderes DV überholen kann und die Reihenfolge der DVs in `driverVehicles` die Vorgänger-Nachfolger-Information darstellt.

```
double breakAx(DriverVehicleW1974* vCurr, DriverVehicleW1974* vLast,
               int posCurr, int posLast, bool* crash);
double breakBx(DriverVehicleW1974* vCurr, DriverVehicleW1974* vLast,
               int posCurr, int posLast);
double follow(DriverVehicleW1974* vCurr, DriverVehicleW1974* vLast,
              DriverVehicleW1974* vLastLast, double timeSpan, int posCurr,
              int posLast, int posLastLast);
double desire(DriverVehicleW1974* vCurr, DriverVehicleW1974* vLast,
              int posCurr, int posLast);
```

Listing 4.8: Deklaration der Verhaltensweisenfunktionen der Klassen `Wiedemann1974`

Die Klasse `Wiedemann1974` implementiert neben der Funktion aus Listing 4.7, Seite 64 die Funktionen aus Listing 4.8. Diese kapseln die vier Verhaltensweisen des Wiedemann-Modells (beschrieben im Abschnitt 2.3.3.3). Die Algorithmusstruktur der Funktion `calculateDriverVehiclePositions(…)` in `Wiedemann1974` entspricht weitgehend der Implementierung in `SimpleFollowTheLeader`.

1. Wird geprüft, ob die DVs in `lane` mit dem Verkehrsmodell korrespondieren.
2. Wird in eine Schleife für die Berechnungen der einzelnen DVs eingetreten.
3. Wird dort gegebenenfalls die Behandlung des CM-DV durchgeführt.
4. Wird ansonsten die Berechnung eines simulierten DVs durchgeführt.
5. Wird nach Abschluss der Aktualisierungen die Sortierung der DVs vorgenommen.

Die Implementierung des vierten Punktes unterscheidet sich natürlich deutlich von der in `SimpleFollowTheLeader`. In `Wiedemann1974` werden an dieser Stelle zunächst die Wahrnehmungsschwellen des DVs durch Aufruf ihrer Funktion `updateWiedemannThresholdVars(…)` aktualisiert. Anschließend wird über den Entscheidungsbaum des Wiedemann-Modells (vergleiche Abbildung 2.5, Seite 22) die entsprechende Verhaltensweise ermittelt. Daraufhin erfolgt der Aufruf der entsprechenden Funktion aus Listing 4.8.
Auch die Implementierung der Reaktionszeit erfolgte gemäß Abschnitt 2.3.3.3. Allerdings wird in BREMSAX und BREMSBX auch dann eine Reaktionsverzögerung von 0.1 Sekunden angenommen, wenn beim direkt vorrausfahrenden DV die Bremslichter angehen oder

`reactionIntensity` größer als 2 ist. Dies geschieht, weil ein Fahrer selbst in dieser Situation immer eine gewisse Zeit zum Reagieren benötigt.
Die eigentliche Berechnung der Beschleunigung erfolgt modellgetreu in der jeweiligen Verhaltensweisenfunktion aus Listing 4.8, Seite 65. Bei der Implementierung der Schätzfehler für Abstand, Geschwindigkeitsdifferenz und Beschleunigung der vorausfahrenden DVs wurde sich für die Variante entschieden, bei der alle drei Größen mit dem jeweiligen Schätzfehler belegt werden und nicht die am Ende berechnete Beschleunigung. Dies ist nach Auffassung des Autors realistischer, da der Fahrer jede der drei Größen verschieden gut einschätzen kann. Wenn Z_N eine $(0.5,0.15)$-normalverteilte Zufallszahl ist, berechnet sich die geschätzte Größe G_S aus ihrem objektiven Wert G_O wie folgt:

$$G_S = G_O \cdot (0.5 + Z_N). \tag{4.1}$$

Die Klasse `Wiedemann1974Extended` implementiert nur die Funktion aus Listing 4.7, Seite 64. Ihre Implementierung ist mit der von `Wiedemann1974` bis auf zwei Punkte identisch. Erstens wird bei der Typüberprüfung der DVs in `lane` auf `DriverVehicleW1974Extended` getestet. Zweitens erfolgt nach Bestimmung der Beschleunigung des DVs in einer der Funktionen aus Listing 4.8, Seite 65 die Implementierung des Trödeleffekts (beschrieben im Abschnitt 2.3.3.4). Bei dem Test, ob das DV ins Trödeln verfällt, wird nicht direkt auf seinen Wert `dawdleProbability` getestet. Der Testwert wird so justiert, dass es bei kürzeren Simulationszykluszeiten unwahrscheinlicher wird ins Trödeln zu verfallen, als bei längeren. Sonst würde bei kürzeren Zeiten bei gleicher Trödelwahrscheinlichkeit häufiger getrödelt werden, da der Trödeltest öfter ausgeführt würde. Der Testwert G berechnet sich wie folgt:

$$G = DB + ((1 - DB) \cdot (1 - LET))^{57}. \tag{4.2}$$

Sollte festgestellt werden, dass das DV zu trödeln beginnt, werden seine Member-Variablen `dawdleing` auf `true` und `dawdleBegin` auf `timeStamp` gesetzt. Bei zukünftigen Simulationsschritten wird dann immer überprüft, ob die Abbruchkriterien für das Trödeln erfüllt sind. Ist das der Fall, wird es abgebrochen und die Member-Variable `dawdleing` des DV auf `false` gesetzt. Andernfalls wird mit dem Trödeln fortgefahren.

4.4.4 Simulationssteuerung

Die Klasse `MainClass` steuert den Ablauf der Simulationsschleife. Sie ist, wie Listing 4.9, Seite 67 zeigt, nicht besonders umfangreich. Der Konstruktor von `MainClass` bekommt mit `settingsName` und `dvName` die Dateinamen der beiden Initialisierungsdateien übergeben. Im

[57] DB entspricht `dawdleProbability` und LET `lastExecutionTime`.

Konstruktor werden zunächst die Werte aus diesen Dateien ausgelesen.[58] Der Parameterwert `eq` wird der Member-Variablen `eventQueue` zugewiesen sowie eine Instanz von `ModelManager` erzeugt und ein Zeiger darauf in der Member-Variablen `mm` gespeichert. Durch Aufruf der Funktion `mainProgrammLoop()` beginnt die Ausführung der Simulationsschleife. Bevor in die Schleife eingetreten wird, erfolgt noch durch Aufruf der Funktion `wakeUpCommunication()` von `mm` die Freischaltung der Kommunikation mit dem Kommunikations- und Visualisierungsmodul.[59] Desweitern wird die lokale Variable `lastExecutionTime`, welche die Simulationszeitschrittdauer enthält, mit `0.01` initialisiert. Da beim ersten Simulationsschritt keine Messung über die Länge des vorausgegangenen Schrittes vorgenommen werden kann, muss diese Variable sinnvoll initialisiert werden. Bei Tests zeigte sich 0.01 als guter Wert. Auch die lokale Variable `runTime`, welche die Gesamtlaufzeit der Simulation festhält, wird sinnvollerweise mit `0` initialisiert. Der eigentliche Ablauf der Simulationsschleife ist dann wie folgt:

1. Bestimmung des aktuellen Zeitpunktes.
2. Aktualisierung von `runTime`.
3. Aufruf der Funktion `calculateDriverVehiclePositions(…)` von `mm` mit den Werten `lastExecutionTime`, aktueller Zeitpunkt und `runTime`. Wodurch die Berechnung der DVs erfolgt.
4. Aufruf der Funktion `sendRelevantCars()` von `mm`. Daraufhin werden die für die Visualisierung benötigten Daten der anzuzeigenden DVs an das Visualisierungsmodul gesendet.[60]
5. Auswertung von `eq` auf neue Events und Reaktion darauf.
6. Bestimmung des aktuellen Zeitpunktes.
7. `lastExecutionTime` bekommt als Wert die Differenz von 5. und 1. zugewiesen.

```
class MainClass
{
private:
   EventQueue *eventQueue;
   ModelManager *mm;
public:
   MainClass(EventQueue* eq, std::string settingsName, std::string dvName);
   void mainProgramLoop(void);
};
```

Listing 4.9: Deklaration der Klasse `MainClass`

Im Konstruktor der Klasse `CalculationModelThread` (siehe Listing 4.10, Seite 68) wird entsprechend des Parameterwertes von `model` eine Instanz eines konkreten Verkehrsmodells, zum Beispiel `Wiedemann1974` erzeugt und ein Zeiger auf diese Instanz in der Member-Variablen `trafficCalculationModel` gespeichert. Die Member-Variable `finish` wird auf `true` und `run` auf `false` gesetzt. Die Funktion `start()` setzt `run` auf `true` und erzeugt eine Instanz von `boost::thread`. Ein Zeiger darauf wird in der Member-Variablen `thread` von

[58] Die Erläuterung wie diese Werte gespeichert und zugänglich gemacht erfolgt in Abschnitt 4.4.6.
[59] Siehe Abschnitt 4.4.5 für ausführliche Erläuterungen.
[60] Siehe Abschnitt 4.4.5 für ausführliche Erläuterungen.

`CalculationModelThread` gespeichert. Durch Listing 4.11 wird letztendlich ein neuer Thread abgespalten, in dem die Funktion `work()` dieser Instanz ausgeführt wird.

```
private:
   TrafficCalculationModel* trafficCalculationModel;
   Lane* lane;
   double timeStamp;
   double lastExecutionTime;
   double carMakerCarVel;
   double carMakerCarLp;
   bool run;
   bool finish;
   boost::thread* thread;
   void work(void);
public:
   static boost::barrier* syncBarrier;
   CalculationModelThread(CalculationModel model =
                                                SIMPLE_FOLLOW_THE_LEADER);
   void start(void);
   void updateValues(Lane* lane, double lastExecutionTime, double timeStamp,
                     double carMakerCarVel = 0, double carMakerCarLp = 0);
   bool isFinished(void);
```

Listing 4.10: Deklaration wichtiger Variablen und Funktionen der Klasse `CalculationModelThread`

```
this->thread = new boost::thread(boost::bind(&CalculationModelThread::work,
                                 this));
```

Listing 4.11: Erzeugung eines `boost::thread` in `start()` von `CalculationModelThread`

In `work()` wird nun, wie in Listing 4.12 ersichtlich, zyklisch die Funktion `calculateDriverVehiclePositions(…)` der Member-Variablen `trafficCalculationModel` ausgeführt und so die Berechnung der DVs vorgenommen. Durch Setzen der Variablen `finish` auf `true` wird ausgedrückt, dass die Berechnungen der DVs abgeschlossen sind. Da nach jedem Berechnungsschritt die Daten der DVs an das Visualisierungsmodul geschickt werden, sind vor einem neuen Berechnungszyklus, die `CalculationModelThread` Threads mit dem Hauptthread zu synchronisieren. Das geschieht über die statische Variable `syncBarrier` vom Typ `boost::barrier`. Durch Aufruf von `wait()` in Listing 4.12 wartet dieser Thread bis die bei der Erzeugung von `syncBarrier` übergebene Anzahl an `wait()` Aufrufen erreicht ist. Erst dann werden alle wartenden Threads wieder freigegeben. Ein Beispiel: Wenn der Verkehr für zwei Fahrspuren simuliert werden soll, wird `syncBarrier` mit dem Wert 3 erzeugt. Das bedeutet, dass die Threads des Programms solange warten bis die jeweiligen Threads der Fahrspuren fertig gerechnet und `wait()` aufgerufen haben sowie der Hauptthread `wait()` ausgeführt hat.

```
void CalculationModelThread::work(void)
{
    while (this->run == true)
    {
        syncBarrier->wait();
        this->trafficCalculationModel->calculateDriverVehiclePositions(
              this->lane, this->lastExecutionTime, this->timeStamp,
              this->carMakerCarVel, this->carMakerCarLp);
        this->finish = true;
    }
}
```

Listing 4.12: Die Funktion `work()` von `CalculationModelThread`

Der Funktion `calculateDriverVehiclePositions(…)` der Variablen `trafficCalculationModel` können die Aufrufparameter durch dieses Konstrukt nicht mehr direkt aus dem Hauptthread heraus übergeben werden. Deswegen sind sie alle als gleichlautende Member-Variablen in der Klasse `CalculationModelThread` vorhanden. Die Werte für diese Member-Variablen werden dann als Parameterwerte übergeben. Ihre Aktualisierung geschieht durch die Funktion `updateValues(…)` aus Listing 4.13.

```
void CalculationModelThread::updateValues(Lane* lane,
                            double lastExecutionTime, double timeStamp,
                            double carMakerCarVel, double carMakerCarLp)
{
    if (this->finish == true)
    {
        this->lane = lane;
        this->timeStamp = timeStamp;
        this->lastExecutionTime = lastExecutionTime;
        this->carMakerCarVel = carMakerCarVel;
        this->carMakerCarLp = carMakerCarLp;
    }
    this->finish = false;
}
```

Listing 4.13: Funktion `updateValues(…)` der Klasse `CalculationModelThread`

Im Konstruktor von `ModelManager` (siehe Listing 4.14, Seite 70) werden im Wesentlichen nur eine Instanz von `Communicator` erzeugt und ein Zeiger darauf der Member-Variablen `communicator` zugewiesen. Abschließend erfolgt der Aufruf der Funktion `initializeLane()`, in der die Instanzen von `Lane` erzeugt und Zeiger darauf in der Member-Variablen `lanes` gespeichert werden. Die Quellen der Fahrspuren liegen dabei 200 Meter vor dem eigentlichen Straßenbeginn und gewähren dadurch neu hinzukommenden DVs eine Einlaufstrecke für das Anpassen ihres Fahrverhaltens. Bei jeder `Lane` Instanz erfolgt ein Aufruf ihrer Funktion `initializeDriverVehicleOnLane()` und somit eine initiale Belegung der Fahrspur mit DVs. Weiterhin ist für jede Instanz von `Lane` eine Instanz von `CalculationModelThread` erzeugt, ein Zeiger darauf in der Member-Variablen `calculationModelThreads` gespeichert und die `start()` Funktion jeder Instanz ausgeführt worden.

Ist kein rein simulierter Verkehr zu erzeugen, so wird in der Funktion `wakeUpCommunication()` zunächst die Funktion `wakeupServer()` von `communicator` aufgerufen und geprüft, ob die Verbindung mit dem Kommunikationsmodul aufgebaut ist. Danach findet der Aufruf der Funktion `waitForVisual()` von `communicator` statt, in welcher der Verbindungsaufbau mit dem Kommunikationsmodul erfolgt. Die Funktion `sendRelevantCars()` ruft die Funktion `sendRelevantCars(vector<Lane*>* lanes)` auf und übergibt für `lanes` die Adresse der Member-Variablen `lanes` von `ModleManager`. Wenn kein rein simulierter Verkehrsstrom erzeugt werden soll, wird in der Funktion `calculateDriverVehiclePositions(…)` zuerst die Funktion `updateCarMakerData()` von `communicator` aufgerufen, bevor die Übertragung der aktuellen Daten des CM-Fahrzeuges vom Kommunikationsmodul stattfindet.

Als nächstes werden für jede Instanz von `Lane` in `lanes` die Funktionen `removeDriverVehicles()` und `addDriverVehicle(…)` ausgeführt. Anschließend erfolgt der

Aufruf der Funktion `wait()` der statischen Variablen `syncBarrier` der Klasse `CalculationModelThread`, wodurch die Threads für die Berechnung der DVs wieder freigeschaltet sind. Nun wird solange gewartet bis in allen `CalculationModelThread` Instanzen die Variable `finish` auf `true` gesetzt ist. Dadurch wird sichergestellt, dass die Berechnung der Verkehrsströme in den abgespaltenen Threads beendet ist.

```
private:
   vector<Lane*> lanes;
   vector<CalculationModelThread*> calculationModelThreads;
   Communicator* communicator;
   void initializeLane(void);
public:
   ModelManager(void);
   void calculateDriverVehiclePositions(double lastExecutionTime,
                                        double timeStamp, double currentRunTime);
   void sendRelevantCars(void);
   void wakeUpCommunication(void);
```

Listing 4.14: Deklaration wichtiger Variablen und Funktionen der Klasse `ModelManager`

Anhand von Abbildung 4.1 soll nun zusammenhängend erklärt werden, wie die parallelisierte Berechnung der Verkehrsströme für zwei Fahrspuren und die Synchronisation der Threads bei Wahl des Wiedemann-Modells ablaufen.

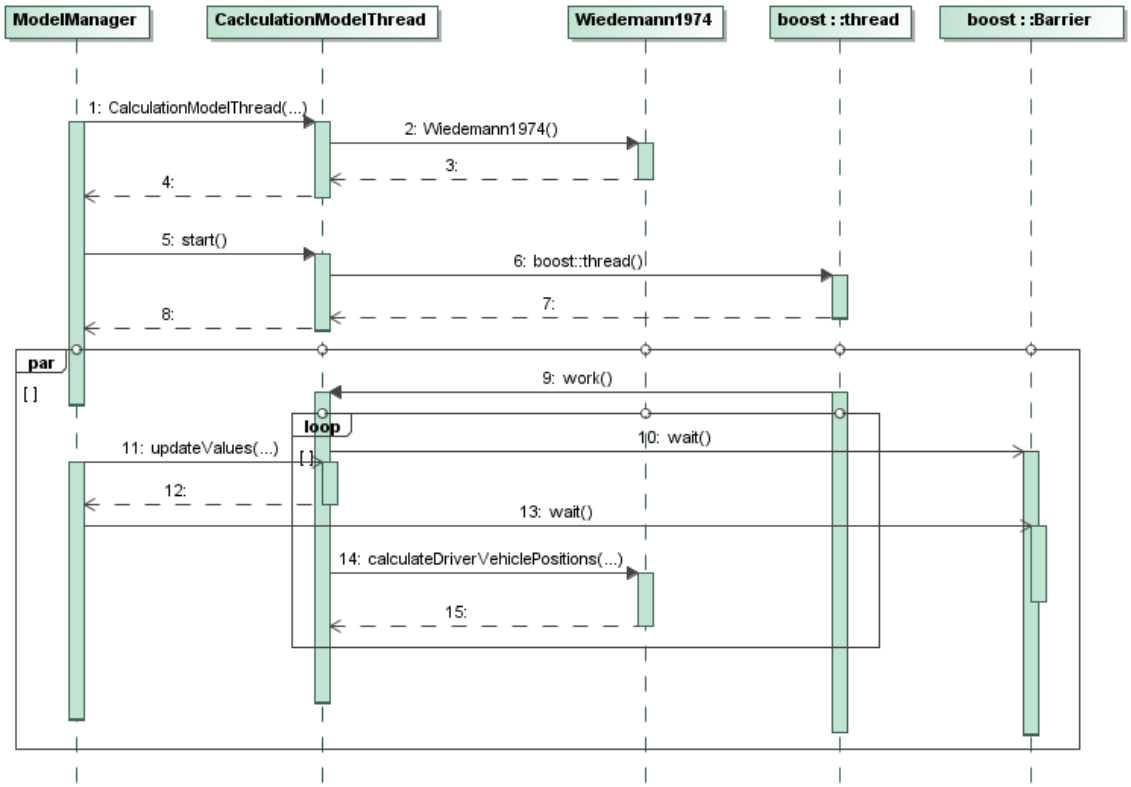

Abbildung 4.1: Ablauf der parallelisierten Berechung mit Wiedemann-Modell

In der Funktion `initializeLane()` einer `ModelManager` Instanz werden die `CalculationModelThread` Instanzen erzeugt (1)[61]. Dabei legen sie jeweils eine Instanz von `Wiedemann1974` an (2). Nach der Konstruktion der Instanzen von `CalculationModelThread` erfolgt bei jeder der Aufruf der Funktion `start()` (5). In ihr wird ein neuer `boost::thread` erzeugt (6). Dieser spaltet nun einen neuen Thread ab und führt die Funktion `work()` darin aus (9). Ab diesem Zeitpunkt beginnt die parallele Abarbeitung. Die abgespaltenen Threads laufen in `work()` gegen die Barriere `boost::barrier syncBarrier` und warten solange, bis die Funktion `wait()` dreimal aufgerufen wurde (10).
Bevor mit der Berechnung der Verkehrsströme begonnen werden kann, müssen die aktuellen Werte für die Aufrufparameter in den `CalculationModelThread` Instanzen gespeichert werden. Das geschieht in der Funktion `calculateDriverVehiclePositions(…)` der `Model-Manger` Instanz durch Aufruf der Funktion `updateCarMakerData()` der `CalculationModelThread` Instanzen (11). Durch den anschließenden Aufruf von `wait()` der statischen Variablen `syncBarrier` von `CalculationModelThread` werden alle Threads wieder freigeschaltet (13). Daraufhin rufen die Berechnungsthreads in `work()` die Funktion `calculateDriverVehiclePositions()` ihrer jeweiligen `Wiedemann1974` Instanz auf (14). Nach der Berechnung laufen sie wieder gegen die Barriere `syncBarrier` (10). Währenddessen wartet die `ModelManager` Instanz solange, bis sämtliche `CalculationModelThread` Instanzen ihre Berechnungen abgeschlossen haben.
Über diesen Mechanismus ist sichergestellt, dass immer nur ein Berechnungszyklus durchgeführt wird. Durch die „Barrier"-Technik ist es zudem unerheblich, welcher der Threads zuerst in Wartestellung geht. Da zwischen den Fahrspuren keine Interaktion stattfindet, müssen auch keine Nebenläufigkeitsprobleme bei der Berechnung beachtet werden.
Durch den Einsatz von boost ist diese Multithreading Implementierung weitestgehend plattformunabhängig.

4.4.5 Netzwerkkommunikation

Die Deklarationen der für den Datenaustausch mit dem Kommunikations- und dem Visualisierungsmodul verwendeten Structs sind in Listing 4.15, Seite 72 ersichtlich. Vom Kommunikationsmodul bekommt das Verkehrssimulationsmodul die benötigten Daten über die Position (`m_sRoad`) und die Geschwindigkeit (`m_velocity`) des CM-Fahrzeuges in der Struktur `CarMaker2TrafficSim` übermittelt. Im Struct `CarVisualEntry` werden die für eine Darstellung im Visualisierungsmodul relevanten Daten übertragen. Diese sind die Position des DVs (`m_sRoad`), der FahrerFarzeug-Typ des DVs (`m_carTyp`) und seine Fahrspur (`m_lane`). Für die Identifizierung der Fahrspuren wurde sich auf das durch Abbildung 3.5, Seite 50 erläuterte Schema verständigt. Beim Fahrzeugtyp erfolgte die Vereinbarung, dass ein DV mit einem Wert von 0 bis 3 durch ein PKW-Modell und bei einem Wert von 4 durch ein LKW-Modell dargestellt werden. Gespeichert werden diese Beschreibungen der DVs in einem Array (`m_data`) in

[61] Die Zahlen in Klammern entsprechen den nummerierten Aufrufen in Abbildung 4.1, Seite 70.

`CarVisualStruct`, welches dann schließlich an das Visualisierungsmodul übertragen wird. Die Beschränkung der Arraygröße auf 200 wird gewählt, weil das Visualisierungsmodul derzeit nicht mehr als 200 Fahrzeuge flüssig darstellen kann.

```
struct CarMaker2TrafficSim
{
   float m_sRoad;
   float m_velocity;
};
struct CarVisualEntry
{
   float m_sRoad;
   unsigned int m_carTyp;
   int m_lane;
};
struct CarVisualStruct
{
   CarVisualEntry m_data[200];
};
```

Listing 4.15: Deklarationen der Structs für den Datenaustausch

Die Klasse `Communicator` implementiert die Kommunikationsfunktionalität mit boost. Dadurch konnte einerseits der für die Implementierung der Sockets benötigte Quellcode von dutzenden Zeilen bei Windows-Sockets auf wenige Zeilen reduziert werden. Der weitaus wichtigere Effekt ist aber andererseits, dass durch den Einsatz von boost-Sockets die Kommunikation weitgehend plattformunabhängig ist. Da die gesamte Kommunikation möglichst schnell vonstattengehen soll und das über das Netzwerk übertragen Datenaufkommen, relativ groß ist, wird UDP statt TCP verwendet. Aufgrund des verwendeten Ethernet-Kabels als Übertragungskanal sind Paketverluste so gut wie ausgeschlossen.
Es wurden drei Selektionsverfahren für die Auswahl der DVs für die Visualisierung implementiert. Beim ersten erfolgt die Selektion aller DVs, aber höchstens die maximal möglichen 200. Bei ihrer Auswahl werden zuerst alle Fahrspuren in Fahrtrichtung des CM-DV von links nach rechts durchgegangen. Anschließend erfolgt die Abarbeitung der Fahrspuren in Gegenfahrtrichtung von rechts nach links.
Die zweite Variante läuft genauso wie die erste ab, nur dass die Selektion der DVs auf einen Bereich (*CullingRange*) um das CM-DV beschränkt ist (siehe Abbildung 4.2, Seite 73). Es werden also nur die DVs ausgewählt, die sich innerhalb von *+CullingRange* vor und *-CullingRange* hinter dem CM-DV befinden.

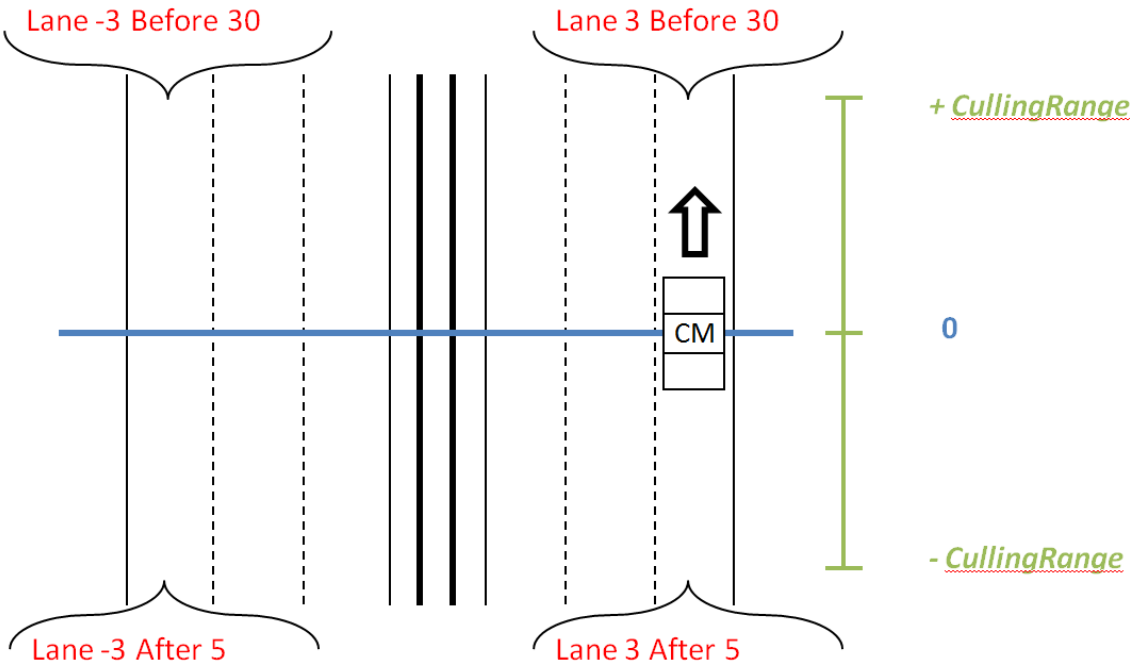

Abbildung 4.2: Beispiel für die Bedeutungen der Auswahlparameter für die Anzeige der DriverVehicle

Die dritte und umfangreichste Art der Selektion soll an Abbildung 4.2 erklärt werden. Grundsätzlich sind nur die DVs selektierbar, die sich innerhalb des durch *CullingRange* bestimmten Auswahlbereiches befinden. Anschließend wird auf jeder Fahrspur eine definierbare maximale Anzahl an DVs vor und hinter dem CM-DV ausgewählt. Im Beispiel aus Abbildung 4.2 würde dies bedeuten, dass auf Fahrspur 3 maximal 30 DVs vor und 5 DVs hinter dem CM-DV für die Visualisierung ausgewählt würden. Analoges gilt für Fahrspur -3 sowie für die restlichen Fahrspuren. Die *CullingRange* und diese Anzahlen pro Fahrspur werden über eines der Initialisierungsdateien übergeben.

```
class Communicator
{
private:
   udp::socket serverSocket;
   udp::socket clientSocket;
   vector<unsigned int> displayedCarNumbers;
   std::string receivedFromIP;
   std::string receivedFromPort;
   std::string sendToPort;
   double currentCarMakerPosition;
   double currentCarMakerVelocity;
   void selectRelevantCars(vector<Lane*>* lanes,
                           CarVisualStruct* carVisualStruct);
public:
   Communicator(std::string receivedFromIP, std::string receivedFromPort,
              , std::string sendToPort);
   bool wakeupServer(void);
   void waitForVisual(void);
   void sendRelevantCars(vector<Lane*>* lanes);
   void updateCarMakerData(void);
   void updateDisplayedCarNumbers(void);
};
```

Listing 4.16: Deklaration wichtiger Variablen und Funktionen der Klasse `Communicator`

Im Konstruktor von `Communicator` (siehe Listing 4.16, Seite 73) werden die UDP-Sockets `serverSocket` und `clientSocket` initialisiert, die Parameterwerte den gleichlautenden Member-Variablen zugewiesen und die Funktion `updateDisplayedCarNumbers()` aufgerufen. In ihr wird die Variable `displayedCarNumbers` mit den Anzahlen der maximal selektierbaren DVs pro Fahrspur vor und hinter dem CM-DV gefüllt.
Die Initialisierung der Kommunikation unter den Modulen findet so statt, dass der Server in der Kommunikationsbeziehung wartet bis er eine Dummy-Nachricht des Clients empfangen hat. Daraufhin geht er in den Normalbetrieb. In der Funktion `wakeupServer()` wird auf dem `serverSocket` eine Dummy-Nachricht an das Kommunikationsmodul gesendet. Während dieses Vorgangs werden verschiedene Fehler abgeprüft. Wenn keine registriert wurden, gibt die Funktion `true` zurück andernfalls `false`. In der Funktion `waitForVisual()` wartet das Verkehrssimulationsmodul solange, bis es eine Dummy-Nachricht vom Visualisierungsmodul auf dem `clientSocket` empfangen hat. Danach ist die Kommunikation initialisiert und der normale Ablauf kann stattfinden.

```
CarMaker2TrafficSim cmReceiveStruct;
char cmVehicle[sizeof(cmReceiveStruct)];
boost::array<char, 1> send_buf = {0};
serverSocket.send_to(boost::asio::buffer(send_buf), receiverEndpoint, 0,
                     errorSending);
serverSocket.receive_from(boost::asio::buffer(cmVehicle,
                          sizeof(cmReceiveStruct)), sender_endpoint,0,
                          errorReceiving);
memcpy(&cmReceiveStruct, &cmVehicle, sizeof(CarMaker2TrafficSim));
```

Listing 4.17: Auszug aus der Funktion `updateCarMakerData()` der Klasse `Communicator`

Die Daten des CM-Fahrzeuges werden in der Funktion `updateCarMakerData()` empfangen. Das geschieht nach dem On-Demand Prinzip. Zuerst wird eine Dummy-Nachricht (`send_buf`) an das Kommunikationsmodul gesendet. Diese signalisiert die Anforderung der aktuellen CM-Fahrzeugdaten. Daraufhin echot das Kommunikationsmodul die Daten zurück. Über die Sockets können grundsätzlich nur Char-Arrays gesendet werden. Deswegen müssen die Structs vor dem Senden in ein Char-Array „gepackt“ und nach dem Empfangen wieder „entpackt“ werden. Dies geschieht durch die Funktion `memcpy(…)`. Sie kopiert byteweise Speicherblöcke von einer Quelle zu einem Ziel. Auf der Seite des Kommunikationsmoduls wurden die aktuelle Position und die aktuelle Geschwindigkeit des CM-Fahrzeuges in einem `CarMaker2TrafficSim` Struct gespeichert. Dieses ist anschließend in ein Char-Array per `memcpy(…)` überführt und dann abgeschickt worden. In der letzten Quellcodezeile in Listing 4.17 ist ersichtlich, dass die empfangen Daten in `cmVehicle` in das Struct `cmReceiveStruct` vom Typ `CarMaker2TrafficSim` kopiert werden. Die aktuelle Position und die aktuelle Geschwindigkeit des CM-Fahrzeuges werden abschließend in den Variablen `currentCarMakerPosition` und `currentCarMakerVelocity` von `Communicator` gespeichert. Beide Werte können über entsprechende Zugriffsfunktionen abgerufen werden. Dieses „packen-entpacken“ Prinzip mit `memcpy(…)` wird von allen Modulen beim Datenaustausch angewendet.
Das Senden der relevanten DV-Daten an das Visualisierungsmodul geschieht in der Funktion `sendRelevantCars(…)`. In ihr wird die Funktion `selectRelevantCars(…)` aufgerufen, die je

nach Selektionsmethode, die Struktur `carVisualStruct` mit den relevanten Daten der ausgewählten DVs füllt. Anschließend wird dieses Struct umgewandelt und an das Visualisierungsmodul gesendet. Die Funktion `selectRelevantCars(…)` implementiert die drei weiter oben beschriebenen Selektionsverfahren. Das erste Verfahren wird benutzt, wenn das Verkehrssimulationsmodul ohne das externe CM-FahrerFahrzeug reinen simulierten Verkehr erzeugt. Soll Verkehr mit dem CM-FahrerFahrzeug erzeugt werden, kann über einen Parameter in einem der Initialisierungsdateien festgelegt werden, ob das zweite oder dritte der weiter oben beschriebenen Verfahren angewendet werden soll.

Im Folgenden soll anhand von Abbildung 4.3 der Kommunikationsablauf des Verkehrssimulationsmoduls erläutert werden.

In der Funktion `mainProgramLoop()` der `MainClass` Instanz wird die Funktion `wakeUpCommunication()` der `ModelManager` Instanz aufgerufen (1)[62]. Anschließend erfolgt durch Aufruf der Funktionen `wakeupServer()` (wenn kein rein simulierter Verkehr erzeugt werden soll) und `waitForVisual()` die Kommunikationinitialisierung mit dem Kommunikationsmodul und dem Visualisierungsmodul (2 und 4). Während des Ablaufs der Simulationsschleife wird die Funktion `calculateDriverVehiclePositions(…)` der `ModelManager` Instanz gerufen (7). In ihr erfolgt die Aktualisierung der Daten des CM-Fahrzeuges, wenn kein rein simulierter Verkehrsfluss erzeugt werden soll (8). Nach der Berechnung der DVs werden die Daten für die Visualisierung gesendet (11 und 12).

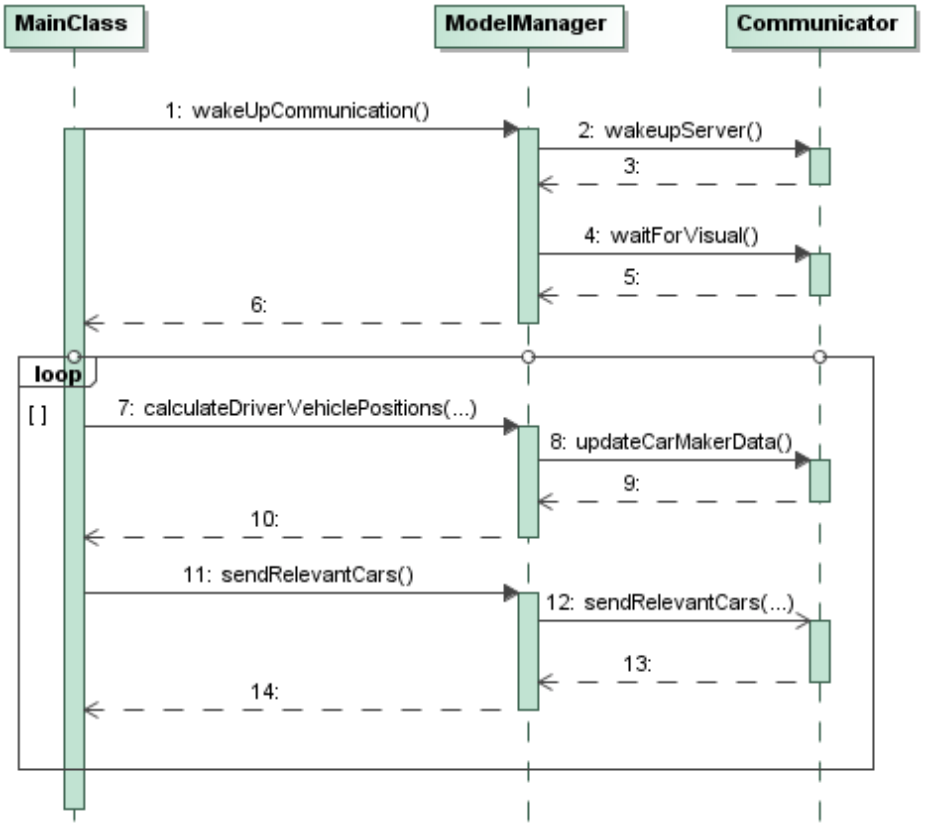

Abbildung 4.3: Kommunikationsablauf

[62] Die Zahlen in Klammern entsprechen den nummerierten Aufrufen in Abbildung 4.3.

4.4.6 Hilfswerkzeuge

Wichtige Funktionalitäten, welche die Implementierung des Verkehrssimulationsmoduls erleichtern beziehungsweise nicht direkt etwas mit der eigentlichen Verkehrssimulation zu tun haben, wie Logging Funktionalität, wurden in separaten Klasse implementiert und sollen in diesem Abschnitt kurz erläutert werden.

RandomGenerator

Bei der Implementierung des Verkehrssimulationsmoduls werden häufig gleichverteilte und normalverteilte Zufallszahlen benötigt. Diese Funktionalität ist in der Klasse `RandomGenerator` gekapselt. Wie aus Listing 4.18 hervorgeht, verfügt `RandomGenerator` über drei statische Funktionen. `getUniformRandom(…)` gibt eine gleichverteilte Zufallszahl zwischen `rangeMin` und `rangeMax` und `getUniformRandomNormalized()` zwischen 0 und 1 zurück. Die Funktion `getNormalRandom(…)` erzeugt eine normalverteilte Zufallszahl um den Erwartungswert `mean` mit einer Standardabweichung von `standardDegression`. Für die Implementierung der Zufallszahlenerzeugung wurde wieder boost verwendet, da es plattformunabhängig ist und die Erzeugung von Zufallszahlen mit verschiedenen Verteilungen ermöglicht.

```
static double getUniformRandom(double rangeMin, double rangeMax);
static double getUniformRandomNormalized(void);
static double getNormalRandom(double mean, double standardDegression);
```

Listing 4.18: Deklaration der Funktionen der Klasse `RandomGenerator`

LoggerFile

Ein Logging Mechanismus ist bei fast jeder Anwendung notwendig, beziehungsweise vorteilhaft. Er wurde in der Klasse `LoggerFile` implementiert. Die geloggten Ereignisse werden auf der Konsole ausgegeben und in einer Datei gespeichert. Über Loglevels kann die Sensitivität des Loggers eingestellt werden. Welches Loglevel verwendet werden soll, ist in einem der Initialisierungsdateien anzugeben (siehe Abschnitt 4.6.1). Der Logger wurde unter Einsatz von boost thread safe implementiert. Der gesamte Programmcode des Verkehrssimulationsmoduls ist mit Logger Aufrufen durchzogen. Je nach eingestelltem Loglevel werden diese Nachrichten dann auch tatsächlich gelogged.

IniFilesHandler

Das Visualisierungsmodul kann über Initialisierungsdateien parametrisiert werden.[63] Die Klasse `IniFilesHandler` (siehe Listing 4.19) liest diese Werte aus, überprüft sie auf formale Korrektheit und stellt sie dem Rest der Anwendung zu Verfügung.
Es wird für jedes der beiden Initialisierungsdateien ein Struct definiert, das die Werte aufnimmt. Das Struct für die Initialisierungsdatei, welches die Anwendung an sich parametrisiert, heißt `SettingsValues` und das Struct für die FahrerFahrzeug-Typen `DriverVehicleValues`.
Durch Aufruf der statischen Funktion `updateSettingsValues(…)` von `IniFilesHandler` werden die ebenfalls statischen Variablen `settingsValues` vom Typ `SettingsValues` und `dvValues` vom Typ `DriverVehicleValues` mit den Parameterwerten gefüllt. Als Aufrufparameter dieser Funktion sind die vollständigen Dateipfade der Initialisierungsdateien zu übergeben. In der Funktion werden die Werte ausgelesen und an den entsprechenden Stellen in den Struct-Variablen von `IniFilesHandler` gespeichert. Zuvor erfolgt eine Überprüfung auf formale Korrektheit. So darf ein Zahlenwert keine Buchstaben enthalten, und eine Prozentzahl muss zwischen 0 und 100 liegen. Wenn eine Initialisierungsdatei nicht gefunden wurde, ein Wert darin nicht angegeben oder ungültiger ist, werden festgelegte Standardwerte in den Variablen gespeichert. Sollte ein solcher Fehler auftreten, erfolgt ein entsprechender Logeintrag in der Datei „IniFilesLogger.log“. Die anderen Klassen des Verkehrssimulationsmoduls erhalten Zugriff auf die Werte der Initialisierungsdateien, indem sie direkt auf die Variablen `settingsValues` und `dvValues` zugreifen.

```
class IniFilesHandler
{
public:
   static SettingsValues settingsValues;
   static DriverVehicleValues dvValues;
   static void updateSettingsValues(std::string filenameSettings,
                                    std::string filenameDV);
};
```

Listing 4.19: Deklaration der Klasse `IniFilesHandler`

Sonstiges

In einer Headerdatei namens „Util.h“ wurden einige weitere nützliche Hilfen implementiert, deren Realisierung in einer Klasse nicht sinnvoll wäre. Diese Datei enthält verschiedene Konvertierungsfunktionen, zum Beispiel einen `string` in einem `wstring`. Außerdem ist hier die Funktion implementiert, welche die Liste der DVs nach einem Berechnungsvorgang mittels QuickSort-Algorithmus sortiert.

[63] Die Erläuterung der Initialisierungsdateien erfolgt in Abschnitt 4.6.

4.5 Start des Verkehrssimulationsmoduls

Das Verkehrssimulationsprogramm wird über das Konsolenprogramm „TrafficSimulationModul.exe“ gestartet. Dieses führt nur die in Listing 4.20 ersichtliche main-Funktion aus. Dem Konsolenprogramm können per Aufrufparameter der vollständige Dateipfad und –name der Initialisierungsdateien übergeben werden. Falls dies nicht passiert, wird angenommen, dass sich diese im aktuellen Verzeichnis befinden und „Settings.ini“ beziehungsweise „DriverVehicles.ini“ heißen. Anschließend wird eine Instanz von `MainClass` erzeugt und die Simulation durch Aufruf der Funktion `mainProgramLoop()` gestartet.

```
int main(int argc, char* argv[])
{
   std::string settingsName = "Settings.ini";
   std::string dvName = "DriverVehicles.ini";
   if(argc == 3)
   {
      settingsName = argv[1];
      dvName = argv[2];
   }
   EventQueue* eq = new EventQueue();
   MainClass *mc = new MainClass(eq, settingsName, dvName);
   mc->mainProgramLoop();
   return 0;
}
```

Listing 4.20: main-Funktion von TrafficSimulationModul.exe

4.6 Initialisierungsdateien

4.6.1 Simulation

Der Standardname für die Initialisierungsdatei der Verkehrssimulation ist „Settings.ini“. Über es können die Netzwerkkommunikation und die Simulation an sich konfiguriert werden. Es gliedert sich in die drei Subsections Network, Traffic und General. Anhand von Listing 4.21, Seite 79 erfolgt die Erläuterung der einzelnen Parameter. Bei unkorrekten Parameterwerten sind jeweils Standardwerte definiert, die in diesem Fall angenommen werden.

```
[Network]
IP=127.0.0.1
VisualisationPort=3000
CarmakerPort=3001
[Traffic]
LaneLength=7852
LaneMode=BOTH
LanesDriveDirection=1
LanesOpposingDriveDirection=1
LaneDDBefore1=100
LaneDDAfter1=30
TrafficVolumeLaneDD1=2000
HeavyLoadTrafficRateDD1=10
LaneODDBefore1=60
LaneODDAfter1=10
TrafficVolumeLaneODD1=1500
HeavyLoadTrafficRateODD1=20
CullingRange=1500
Model=WIEDEMANN1974EXTENDED
UseCarMaker=True
UseDisplaySelection=True
[General]
Loglevel=ERR
```

Listing 4.21: Beispiel einer Initialisierungsdatei für die Simulation

Die Subsection *Network* wird in dieser Initialisierungsdatei durch den Tag `[Network]` eingeleitet. Hier stehen die Parameter, die für die Kommunikation im Verteilten System mit dem Kommunikationsmodul und dem Visualisierungsmodul wichtig sind. Es umfasst die drei Parameter `IP`, `CarmakerPort` und `VisualisationPort`. Ihre Bedeutung wird im Folgenden erläutert.

IP

Dieser Parameter bezeichnet die IPv4 Adresse des Rechners, auf dem das Kommunikationsmodul ausgeführt wird.
Standardwert: 127.0.0.1

VisualisationPort

Dieser Parameter bezeichnet den Port des Rechners, auf dem das Visualisierungsmodul ausgeführt wird.
Standardwert: 3000

CarmakerPort

Dieser Parameter bezeichnet den Port des Rechners, auf dem das Kommunikationsmodul ausgeführt wird.
Standardwert: 3001

Die Subsection *Traffic* wird in dieser Initialisierungsdatei durch den Tag `[Traffic]` eingeleitet. Die unter ihr aufgeführten Parameter definieren die Einstellungen für die Art und Weise sowie den Umfang der simulierten Verkehrsströme und die Anzeige der simulierten Fahrzeuge. Die Anzahl der Parameter in der Subsection *Traffic* hängt von der Anzahl der Fahrspuren mit simuliertem Verkehr ab. Diese dynamischen Parameter sind `LaneDDBefore<N>`, `LaneDDAfter<N>`, `LaneODDBefore<M>`, `LaneODDAfter<M>`, `TrafficVolumeLaneDD<N>`, `TrafficVolumeLaneODD<M>`, `HeavyLoadTrafficRateDD<N>` und `HeavyLoadTrafficRateODD<M>`. N ist eine Ganzzahl im Intervall [1, Anzahl an Fahrspuren in Fahrtrichtung des CM-DV]. Der Bezug zur Fahrspur ist dabei [linkeste Fahrspur in Fahrtrichtung des CM-DV, rechteste Fahrspur in Fahrtrichtung des CM-DV]. Die Ganzzahl M verläuft im Intervall [1, Anzahl an Fahrspuren in Gegenfahrtrichtung des CM-DV]. Der Bezug zur Fahrspur ist dabei [rechteste Fahrspur in Gegenfahrtrichtung des CM-DV, linkeste Fahrspur in Gegenfahrtrichtung des CM-DV] (vergleiche Abbildung 4.4, Seite 82).
Die Bedeutung der dynamischen Parameter und ihr Zusammenwirken werden im weiteren Verlauf dieses Abschnittes beschrieben, ebenso die Bedeutung der statischen Parameter `LaneLength`, `LaneMode`, `LanesDriveDirection`, `LanesOpposingDriveDirection`, `CullingRange`, `Model`, `UseCarMaker` und `UseDisplaySelection`.

LaneLength

Dieser Parameter bezeichnet die Länge der Straße in Metern. Alle Fahrspuren wird diese Länge zugeordnet.
Standardwert: 8000

LaneMode

Dieser Parameter gibt die Fahrtrichtung des zu erzeugenden Verkehrs an. Möglich sind beide Richtungen, nur in Fahrtrichtung des CM-DVs und nur in Gegenfahrtrichtung des CM-DVs. Bei nicht korrespondierenden Werten dieses Parameters mit den Parametern `LanesDriveDirection` und `LanesOpposingDriveDirection` hat der Wert von `LaneMode` Vorrang.
Zulässige Werte für diesen Parameter sind: BOTH, DRIVE_DIRECTION und OPPOSING_DRIVE_DIRECTION.
Standardwert: BOTH

LanesDriveDirection

Dieser Parameter gibt die Anzahl an Fahrspuren in Fahrtrichtung des CM-DVs an, auf denen ein Verkehrsfluss simuliert werden soll.
Standardwert: 1

LanesOpposingDriveDirection

Dieser Parameter gibt die Anzahl an Fahrspuren in Gegenfahrtrichtung des CM-DVs an, auf denen ein Verkehrsfluss simuliert werden soll.
Standardwert: 1

CullingRange

Dieser Parameter bezeichnet den Abstand in Metern, in dem jeweils vor und hinter der Position des CM-DVs Fahrzeuge für die Anzeige im Visualisierungsmodul ausgewählt werden. (vergleiche Abbildung 4.4, Seite 82)
Standardwert: 1500

Model

Dieser Parameter gibt das Verkehrsmodell an, auf dessen Basis der Verkehr auf den jeweiligen Fahrspuren erzeugt wird.
Zulässige Werte für diesen Parameter sind: SIMPLE_FOLLOW_THE_LEADER, WIEDEMANN1974, WIEDEMANN1974EXTENDED.
Standardwert: SIMPLE_FOLLOW_THE_LEADER

UseCarMaker

Dieser Parameter gibt an, ob die Simulation mit dem CM-DV durchgeführt werden soll. Zulässige Werte für diesen Parameter sind: True, true, False, false.
Standardwert: true

UseDisplaySelection

Dieser Parameter gibt an, ob die Auswahl der FahrerFahrzeuge für die Anzeige nach der dritten Variante (Wert: true) oder der zweiten Variante (Wert: false) aus Abschnitt 4.4.5 durchgeführt werden soll. Zulässige Werte für diesen Parameter sind: True, true, False, false.
Standardwert: true

LaneDDBefore<N> und LaneODDBefore<M>

Diese Parameter bestimmen, wie viele DVs maximal vor der Position des CM-DV auf den jeweiligen Fahrspuren angezeigt werden sollen. Dies gilt allerdings nur für DVs, die sich innerhalb des durch den Parameter `CullingRange` definierten Bereiches befinden.

In Abbildung 4.4 ist ein Beispiel für die Bedeutung dieser Parameter angegeben. Auf der rechten Fahrspur in Fahrtrichtung des CM-DV würden maximal 30 DVs vor ihm angezeigt, auf der linken in Gegenfahrtrichtung ebenfalls 30.
Standardwert: 50

LaneDDAfter<N> und LaneODDAfter<N>

Diese Parameter geben an, wie viele DVs maximal hinter der Position des CM-DV auf ihren jeweiligen Fahrspuren angezeigt werden sollen. Das gilt allerdings nur für DVs, die sich innerhalb des durch den Parameter `CullingRange` definierten Bereiches befinden. In Abbildung 4.4 ist ein Beispiel für die Bedeutung dieser Parameter angegeben. Auf der rechten Fahrspur in Fahrtrichtung des CM-DV würden maximal 5 DVs hinter ihm angezeigt, auf der linken in Gegenfahrtrichtung ebenfalls 5.
Standardwert: 10

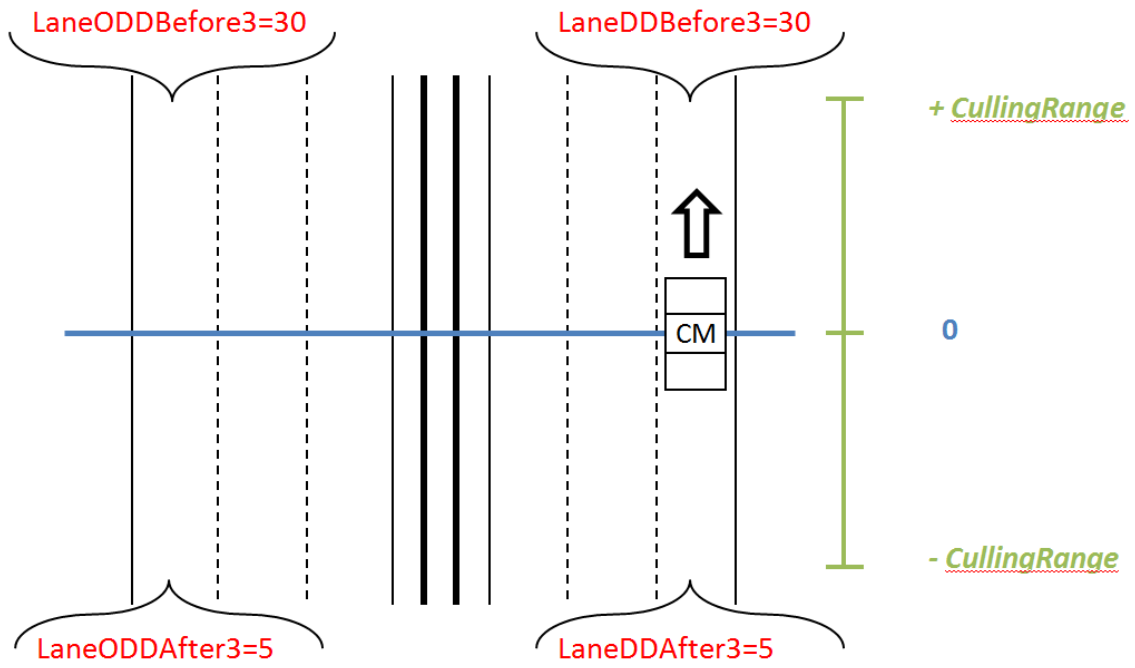

Abbildung 4.4: Beispiel für die Anzeigeparameter

TrafficVolumeLaneDD<N> und TrafficVolumeLaneODD<N>

Diese Parameter geben die Verkehrsstärke in Fz/h für die jeweiligen dazugehörigen Fahrbahnen an.
Standardwert: 1000

HeavyLoadTrafficRateDD und HeavyLoadTrafficRateODD

Diese Parameter geben den Anteil des Schwerlastverkehrs am Gesamtverkehrsaufkommen in Prozent für die jeweiligen dazugehörigen Fahrbahnen an. Der Wertebereich dieses Parameters geht von 0 bis 100.
Standardwert: 20

Die Subsection *General* wird in dieser Initialisierungsdatei durch den Tag `[General]` eingeleitet. Hier stehen die Parameter mit einer allgemeinen bzw. sonstigen Bedeutung für das Verkehrssimulationsmodul. Es umfasst den Parameter `Loglevel`.

Loglevel

Dieser Parameter gibt die Sensitivität des Loggers des Verkehrssimulationsmoduls an. Der Tabelle 4.1 sind die zulässigen Parameterwerte und deren Bedeutung entnehmbar.
Standardwert: ERR

Tabelle 4.1: Zulässige Parameterwerte von `Loglevel` und deren Bedeutung

Parameterwert	Bedeutung
DEBUG	Alle Log-Einträge, die unter INFO geschrieben werden und zahlreiche zusätzliche Angaben, die für die Entwicklungsarbeit von Bedeutung sind. Beispielsweise Charakteristika jedes DVs nach jedem Berechnungsschritt. Unter DEBUG werden sehr schnell sehr große Logfiles erzeugt.
INFO	Alle Log-Einträge, die unter WARN geschrieben werden. Zusätzlich werden Informationen, wie die verstrichene Simulationszeit, die Berechnungszeit eines Simulationsschrittes und die Anzahl der für die Anzeige ausgewählten Fahrzeuge, gelogged.
WARN	Alle Log-Einträge, die unter ERR geschrieben werden. Zusätzlich werden Benachrichtigungen gelogged, falls zum Beispiel keine Initialisierungsdatei oder keine Verbindung zum Kommunikationsmodul vorhanden sind.
ERR	Es werden Fehler gelogged, die während der Laufzeit der Simulation auftreten können. Unter anderem dann, wenn es einen Unfall zwischen zwei Fahrzeugen auf einer Fahrspur gibt, Bereichsüberschreitungen von Variablen stattfinden oder die Kommunikation zum Visualisierungsmodul abbricht.
NONE	Es werden keine Log-Einträge vorgenommen.

Wie aus diesem Abschnitt deutlich wird, kann das Verkehrssimulationsmodul vielfältig und detailliert parametrisiert werden und so ein relativ großes Spektrum an Strecken und Verkehrssituationen nachbilden.

4.6.2 FahrerFahrzeuge

Der Standardname für die Initialisierungsdatei der FahrerFahrzeug-Typen ist „DriverVehicles.ini“. Darüber können vor allem die Geschwindigkeitsverteilung und die Auftrittshäufigkeit der verschiedenen Typen konfiguriert werden. Es gliedert sich in eine statische Subsection *META* und dynamisch in viele weitere Subsections *CAR<N>* und *TRUCK<M>*. Deren Anzahl hängt von der eingestellten Anzahl an unterschiedlichen PKW- und LKW-Typen ab. Anhand von Listing 4.22 sollen im weiteren Verlauf dieses Abschnitts die einzelnen Parameter erläutert werden. Für jeden Parameter ist ein Standardwert angegeben, der im Fall eines unkorrekten Wertes angenommen wird.

```
[META]
NumberCars=3
NumberTrucks=1
[CAR1]
Length=5.4
DesiredVelocityMean=110
DesiredVelocityDegression=6
ProbabilityRate=20
[CAR2]
Length=5.1
DesiredVelocityMean=110
DesiredVelocityDegression=6
ProbabilityRate=30
[CAR3]
Length=6.0
DesiredVelocityMean=110
DesiredVelocityDegression=6
ProbabilityRate=30
[TRUCK1]
Length=13.4
DesiredVelocityMean=90
DesiredVelocityDegression=1.67
ProbabilityRate=20
```

Listing 4.22: Beispiel einer Initialisierungsdatei für die FahrerFahrzeug-Typen

Die Subsection *META* wird in dieser Initialisierungsdatei durch den Tag `[META]` eingeleitet. Die unter ihr aufgeführten Parameter geben die Anzahl verschiedener PKW- und LKW-Typen in dieser Initialisierungsdatei an. Diese Subsection umfasst die Parameter `NumberCars` und `NumberTrucks`.

NumberCars

Dieser Parameter gibt die Anzahl an PKW-FahrerFahrzeug-Typen an, die in dieser Initialisierungsdatei definiert werden.
Standardwert: 4

NumberTrucks

Dieser Parameter gibt die Anzahl an LKW-FahrerFahrzeug-Typen an, die in dieser Initialisierungsdatei definiert werden.
Standardwert: 1

Die Anzahl der Subsections *CAR<N>* richtet sich nach dem Wert von `NumberCars`. Die Ganzzahl N verläuft dabei in dem Intervall [1, `NumberCars`]. Sind weniger *CAR<N>* Subsections definiert als in `NumberCars` angegeben, so werden für die noch fehlenden Standardwerte angenommen. Wenn mehr definiert sind, werden die Subsections, bei denen N größer als `NumberCars` ist, ignoriert. Analoges gilt für *TRUCK<M>* bezogen auf `NumberTrucks`.
Die einzelnen Subsections werden durch die Tags `[CAR<N>]` oder `[TRUCK<M>]` eingeleitet. Beide Subsections umfassen die gleichen Parameter `Length`, `DesiredVelocityMean`, `DesiredVelocityDegression` und `ProbabilityRate`.

Length

Dieser Parameter gibt die Länge des FahrerFahrzeug-Typs in Metern an.
Standardwert: 5.5

DesiredVelocityMean

Dieser Parameter gibt den Erwartungswert in km/h an, um den die Wunschgeschwindigkeit (bei den Wiedemann-Modellen) beziehungsweise die initiale Geschwindigkeit (beim Fahrzeugfolgemodell) der DVs normalverteilt ist.
Standardwert: 110

DesiredVelocityDegression

Dieser Parameter gibt die Standardabweichung in km/h an, um die die Wunschgeschwindigkeit (bei den Wiedemann-Modellen) beziehungsweise die initiale Geschwindigkeit (beim Fahrzeugfolgemodell) der DVs streut.
Standardwert: 6

ProbabilityRate

Dieser Parameter gibt die Häufigkeit in Prozent an, mit der ein DV dieses Typs im Verkehrsstrom auftaucht.
Standardwert: 100/(`NumberCars` + `NumberTrucks`)

Die Standardwerte für `DesiredVelocityDegression` und `ProbabilityRate` sind aus [Erle07] entnommen und basieren auf den Untersuchungen von [BrWeWu00]. Durch die freie Skalierbarkeit der Anzahl der FahrerFahrzeug-Typen mit ihren Geschwindigkeitsverteilungen und Auftrittshäufigkeiten ist ein hoher Freiheitsgrad beim Einsatz des Verkehrssimulationsmoduls gegeben.

4.7 Grafische Benutzeroberfläche

Zur Erleichterung der Bedienung des Verkehrssimulationsmoduls wurde eine GUI mit C++/CLI implementiert. Bei einer Portierung des Moduls auf eine andere Plattform müsste diese Modulkomponente weitgehend neu entwickelt werden. Da die GUI aber nicht zwingend für die Verkehrssimulation benötigt wird, ist das von eher geringer Bedeutung. Der Hauptzweck der GUI besteht darin, relativ einfach, korrekte Initialisierungsdateien zu erzeugen. Desweiteren können über sie der Start und die Terminierung der Verkehrssimulation erfolgen. Es wurde versucht, die GUI optisch klar und logisch zu strukturieren, um eine möglichst einfache und intuitive Bedingung zu ermöglichen.
Anhand von Abbildung 4.5, Seite 87 und Abbildung 4.6, Seite 88 soll zunächst der strukturelle Aufbau der GUI erläutert werden.
Den größten Teil der Oberfläche nimmt das mit 1 gekennzeichnete *TabPannel* ein. In dem Tab „Lanes“ (1)[64] können die Fahrspur und im Tab „Driver Vehicles“ (5) die FahrerFahrzeug-Typen bezogenen Einstellungen vorgenommen werden. Im Panel „Network“ (2) erfolgt die Angabe der Netzwerkverbindungseinstellungen. Das Panel „Traffic Model“ (3) dient der Festlegung des zu verwendenden Verkehrsmodells. In der Button-Leiste (4) sind die verschiedenen ausführbaren Aktionen definiert.
Im Tab „Lanes“ können zunächst allgemeine Einstellungen für die Fahrspuren vorgenommen werden, wie ihre Anzahl (1.1) oder ihre Längen in Metern (1.2). In 1.3 erfolgt die Auswahl, ob das detaillierte Selektionsverfahren für die Anzeige der FahrerFahrzeuge zu verwenden ist. In den restlichen Feldern können die pro Fahrspur spezifischen Eigenschaften festgelegt werden. Die über die GUI definierte Anzahl von Fahrspuren ist auf acht beschränkt. Diese Anzahl wird in den allermeisten Fällen mehr als ausreichend sein. Sollte dennoch einmal mehr Fahrspuren benötigt werden, sind diese „per Hand“ in die Initialisierungsdatei einzutragen. Für jede Fahrspur ist einstellbar, ob der Verkehrsfluss auf ihr in Fahrt- oder Gegenfahrtrichtung des

[64] Die Zahlen in Klammern entsprechen den Nummerierungen in Abbildung 4.5, Seite 87 und Abbildung 4.6, Seite 88.

externen CM-Fahrzeuges fließen soll (1.4). Sofern 1.3 aktiviert ist, kann in 1.5 und 1.6 die maximale Anzahl an Fahrzeugen angegeben werden, die vor beziehungsweise hinter dem CM-Fahrzeug auf dieser Fahrspur anzuzeigen sind. In 1.7 erfolgt die Festlegung der Verkehrsstärke für diese Spur und in 1.8 der Anteil des Schwerlastverkehrs in Prozent.
Im Panel „Network“ (2) werden die IP-Adresse und der Port des Kommunikationsmoduls sowie der Port des Visualisierungsmoduls angegeben.
Das Panel „Traffic Model“ (3) dient der Einstellung des zu verwendenden Verkehrsmodells und der Angabe, ob das externe CM-Fahrzeug mit berücksichtigt wird.
Für die Erzeugung einer gültigen Simulations-Initialisierungsdatei genügt es, die Maske aus Abbildung 4.5 auszufüllen.

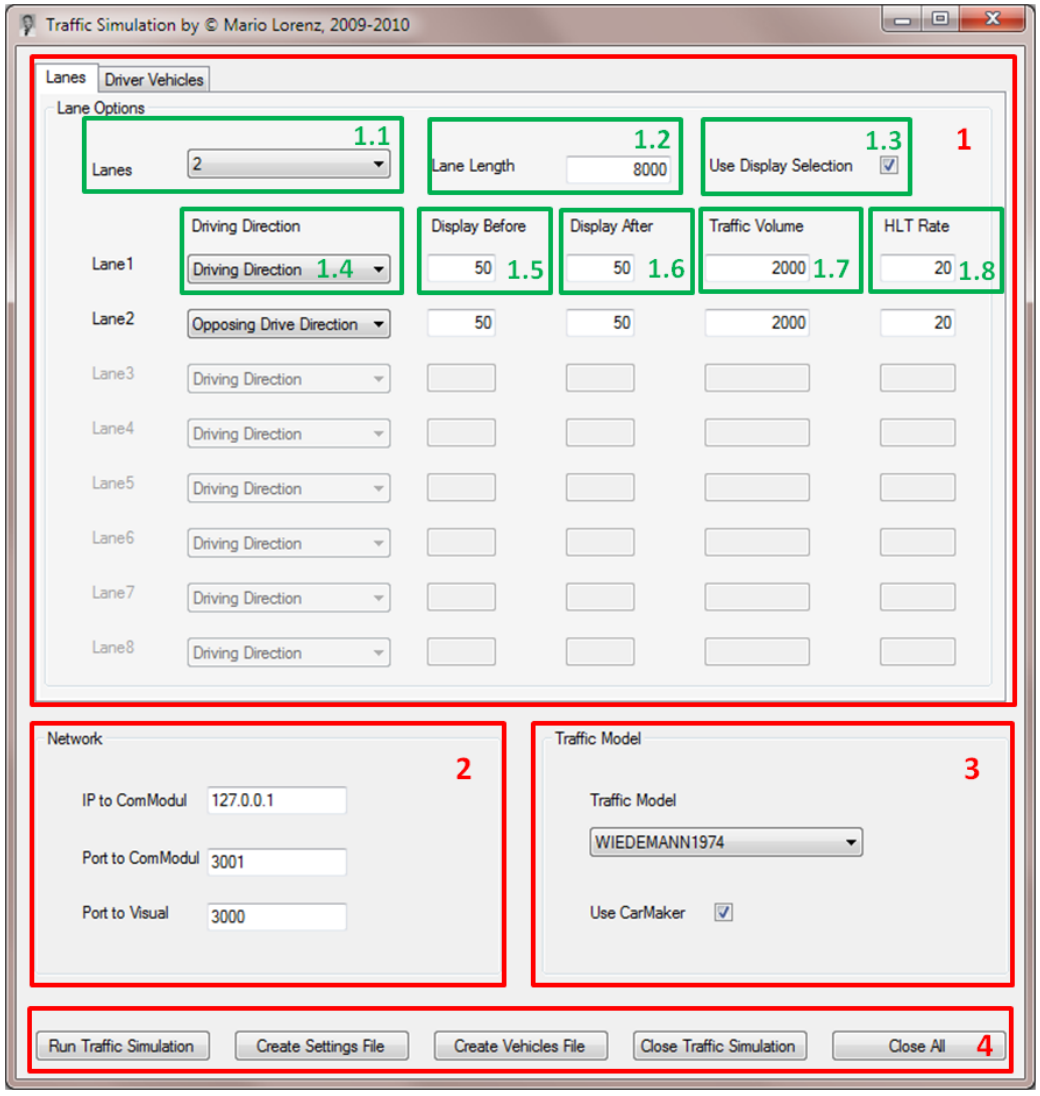

Abbildung 4.5: Aufteilung der grafischen Benutzeroberfläche

Um eine gültige FahrerFahrzeug-Initialisierungsdatei zu erzeugen ist es ausreichend, den Tab „Driver Vehicles“ (5) auszufüllen.
In ihm ist zunächst anzugeben, wie viele FahrerFahrzeug-Typen definiert werden sollen (5.1). Auch hier wurde die Anzahl auf acht beschränkt. Müssten mehr FahrerFahrzeug-Typen

definiert werden, kann dies wieder direkt „per Hand“ in der entsprechenden Initialisierungsdatei geschehen. Für jedes FahrerFahrzeug wird eingestellt, ob es sich um einen PKW (CAR) oder LKW (TRUCK) (5.2) handelt. Dann erfolgt die Definition seiner Länge (5.3) und die Angabe der Geschwindigkeitsverteilung (5.4 und 5.5). Abschließend ist unter 5.6 die Auftrittshäufigkeit für diesen FahrerFahrzeug-Typ anzugeben.

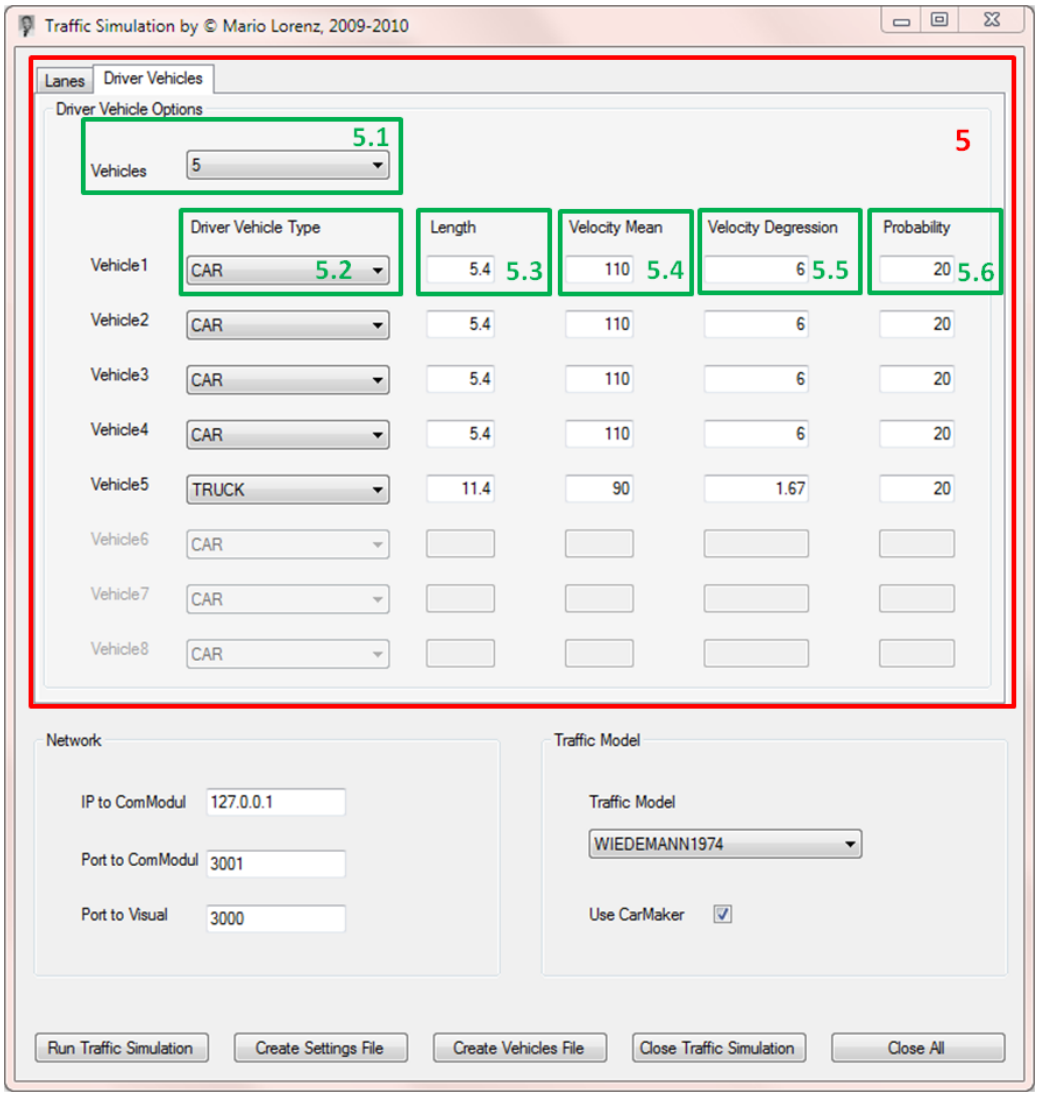

Abbildung 4.6: Grafische Benutzeroberfläche mit Einstellungen für FahrerFahrzeug-Typen

Wurden sämtliche Felder ausgefüllt, können über die Buttons verschiedene Aktionen ausgeführt werden. Die Buttons „Create Settings File“ und „Create Vehicles File“ erzeugen jeweils die Simulations-Initialisierungsdatei beziehungsweise die FahrerFahrzeug-Initialisierungsdatei im Ausführungsverzeichnis der GUI. Durch Drücken des Buttons „Run Traffic Simulation“ werden zunächst die beiden Initialisierungsdateien mit den in der GUI eingetragenen Werten in ihrem Ausführungsverzeichnis erzeugt. Anschließend wird das Konsolenprogramm zum Start der Verkehrssimulation aufgerufen. Dieses muss sich ebenfalls im selben Verzeichnis wie die GUI befinden. Über den Button „Close Traffic Simulation“ wird das Konsolenprogramm der

Verkehrssimulation terminiert. Durch Drücken des Buttons „Close All“ erfolgt die zusätzliche Beendigung der GUI.
Die GUI ist so implementiert, dass nur dann Aktionen ausgeführt werden, wenn alle benötigten Daten vorhanden und formal korrekt sind. Ist das nicht der Fall, wird der Nutzer durch Fehlermeldungen darauf hingewiesen.

4.8 Zeitverhalten

In diesem Abschnitt soll erläutert werden, wovon die Zeitdauer eines Berechnungszyklus‘ abhängt und wie schnell die Berechnung abläuft. Desweiteren wird mit empirischen Daten das Zutreffen der im Abschnitt 3.4 getroffenen Aussage bezüglich des Fehlerausgleichs der Berechnungszykluszeit unterstrichen.
Es wurden drei aufeinanderfolgende Messungen mit jeweils einem anderen Verkehrsmodell bei sonst gleichen Bedingungen (siehe Tabelle 4.2) durchgeführt. Gemessen wurden die Zeitdauer der Berechnungszyklen und die Anzahl der verwalteten FahrerFahrzeuge. Die Rohdaten liegen dieser Masterarbeit auf der CD im Verzeichnis *.\Tests\Simulationszeitdauertests* bei.

Tabelle 4.2: Versuchsbedingungen der Zeitmessungen

Parameter	Parameterwert
Streckenlänge	8000 m
Verkehrsstärke	2500 FZ/h
Fahrspuren	2
Hardware	Windows 7 Professional 64-Bit; Intel Core 2Duo E8500 3,16 GHz; 1333 MHz FSB; 4 GB RAM
Prozessorkerne	2

Anhand von Tabelle 4.3, Seite 90 sollen nun die weiteren Ausführungen erläutert werden.
Die Messdaten wurden jeweils über einen Zeitraum von knapp 30 Minuten aufgezeichnet. Die durchschnittliche Berechnungszykluszeit ist bei den beiden Wiedemann-Modellen mit rund 8 Millisekunden nahezu gleich, während sie beim Fahrzeugfolgemodell mit rund 7 Millisekunden etwas geringer ist. Eine genauere Betrachtung der Rohdaten zeigt, dass bei den Messungen mit den Wiedemann-Modellen die ersten Zykluszeiten ungefähr 1.5 Millisekunden betragen und beim Fahrzeugfolgemodell ungefähr 0.75 Millisekunden. Zum Ende der Messungen liegen sie bei den Wiedemann-Modellen bei ungefähr 24.5 Millisekunden und beim Fahrzeugfolgemodell bei ungefähr 26 Millisekunden.
Wie ist dieser starke Anstieg zu erklären?
Ein Vergleich von Abbildung A.1, Seite IX, Abbildung A.2, Seite X und Abbildung A.3, Seite X zeigt eindeutig, dass die Berechnungszykluszeiten proportional zu der Anzahl der zu berechnenden FahrerFahrzeuge sind. Weiterhin lässt sich anmerken, dass bei knapp 2000 verwalte-

ten FahrerFahrzeugen noch akzeptable Zykluszeiten erreicht werden, die eine flüssige Visualisierung ermöglichen.

Maßgebliche Geschwindigkeitsunterschiede zwischen den Modellen lassen sich kaum erkennen. So liegen die Berechnungszykluszeiten bei 800 verwalteten FahrerFahrzeugen mit circa 19 Millisekunden beim Fahrzeugfolgemodell und circa 21 Millisekunden bei den Wiedemann-Modellen nur unwesentlich auseinander. Aus diesen Erläuterungen und der durchschnittlichen Zykluszeit lässt sich aber erkennen, dass das Fahrzeugfolgemodell das schnellste Modell und das erweiterte Wiedemann-Modell nur minimal langsamer als das ursprüngliche Wiedemann-Modell ist (vergleich Tabelle 4.3). Erklärbar ist das durch den ansteigenden Aufwand, der pro FahrerFahrzeug für die Berechnung durchgeführt werden muss.[65]

Bei den Zykluszeiten treten immer wieder starke Spitzen auf, die sich durch das Scheduling des Betriebssystems erklären lassen. Da das Verkehrssimulationsmodul bei den Tests mit drei Threads auf nur zwei Prozessorkernen lief, kommt es dadurch wahrscheinlich zu einem gewissen Ausbremsen. Optimalerweise sollte das Modul auf einem Rechner ausgeführt werden, der mindestens $Anzahl\ Fahrspuren+1$ Kerne hat.

Tabelle 4.3: Ergebnisse der Messungen

Bezeichnung	Fahrzeugfolge-M.	Wiedemann-M.	Erw. Wiedemann-M.
Messzeitraum (min.)	29.9175	29.8110	29.9380
Messpunkte (Anzahl)	259758	226089	215067
Mittelwert (s)	0.0069105	0.007911339	0.008352216
Mittelwert-Diff (s)	$6.8861 \cdot 10^{-8}$	$6.5594 \cdot 10^{-8}$	$7.2862 \cdot 10^{-8}$

Aus den Messergebnissen der Berechnungszykluszeiten wurde eine weitere Reihe gebildet. Dabei erfolgt für alle Messpunkte die Differenzbildung aus der Berechnungszykluszeit am Messpunkt $n+1$ mit der am Messpunkt n. Damit soll überprüft werden, ob sich die Fehler, die von Zyklus zu Zyklus auftreten, wirklich aufheben. Die Mittelwerte (Tabelle 4.3: Mittelwert-Diff) dieser Reihen liegen bei den drei Messungen zwischen 65,6 und 72,9 Nanosekunden. Dies unterstreicht die in Abschnitt 3.4 getroffene Aussage, dass sich die Fehleinschätzungen bezüglich der Berechnungszykluszeit im nächsten Zyklus ausgleichen.

[65] Dass zum Ende der Messungen das Fahrzeugfolgemodell eine längere Berechnungszykluszeit aufweist erklärt sich dadurch, dass es zu diesem Zeitpunkt circa 270 FahrerFahrzeuge mehr verwaltet als die beiden Wiedemann-Modelle.

5 Vergleichende Bewertung der implementierten Verkehrsmodelle

5.1 Einleitung

In diesem Kapitel werden die im Verkehrssimulationsmodul implementierten Verkehrsmodelle miteinander verglichen und bewertet. Zunächst soll im Abschnitt 5.2 anhand der Trajektorien (Bewegungspfade) der FahrerFahrzeuge die Realitätsnähe der Verkehrsmodelle untersucht werden.
Eine individuelle Bewertung des Fahrens im Verkehrsstrom bezüglich der Realitätsnähe durch Probanden und ein Vergleich der drei verwendeten Verkehrsmodelle wird im Abschnitt 5.3 gegeben

5.2 Simulierter Verkehr

5.2.1 Vorbemerkungen

Um den erzeugten Verkehr der drei implementierten Verkehrsmodelle zu vergleichen, wurde mit jedem Modell ein Simulationsdurchlauf ausgeführt. In Tabelle 5.1, Seite 92 stehen die Randbedingungen, die bei allen drei Durchläufen konstant geblieben sind. Für jeden Durchlauf wurde lediglich das Verkehrsmodell geändert. Die Aufzeichnungen der FahrerFahrzeug-Positionen erfolgten in einem 3 Sekunden Intervall über einen Zeitraum von 30 Minuten. Die Erstellung der grafischen Darstellungen der Trajektorien erfolgte mit der mathematischen Software MATLAB. Die Rohdaten liegen dieser Masterarbeit auf der CD im Verzeichnis .*Tests**SimulationstestsVerkehr* bei.

Tabelle 5.1: Versuchsbedingungen der Trajektorienmessungen

Parameter	Parameterwert
Verkehrsstärke	2500 Fz/h
Fahrspuren	2
SV-Anteil	20 %
SV-Geschwindigkeitsverteilung	Erwartungswert $\mu = 90\ km/h$ Standardabweichung $\sigma = 1.67\ km/h$
PKW-Geschwindigkeitsverteilung	Erwartungswert $\mu = 110\ km/h$ Standardabweichung $\sigma = 6\ km/h$
Hardware	Windows 7 Professional 64-Bit; Intel Core 2Duo E8500 3,16 GHz; 1333 MHz FSB; 4 GB RAM
Prozessorkerne	2

5.2.2 Fahrzeugfolgemodell

Abbildung 5.1, Seite 93 zeigt den Verlauf der Trajektorien der FahrerFahrzeuge über den gesamten Messzeitraum. Schon in dieser sehr grob aufgelösten Darstellung lässt sich die Uniformität der FahrerFahrzeug-Bewegungen erahnen. Sie fahren alle in einer riesigen Kolonne mit geringen Abständen hintereinander her. Trotz der unterschiedlichen initialen Geschwindigkeiten entstehen kein größeren Lücken im Verkehrsstrom, da sich schon nach kurzer Zeit alle FahrerFahrzeuge an die Geschwindigkeit des führenden FahrerFahrzeuges angepasst haben.

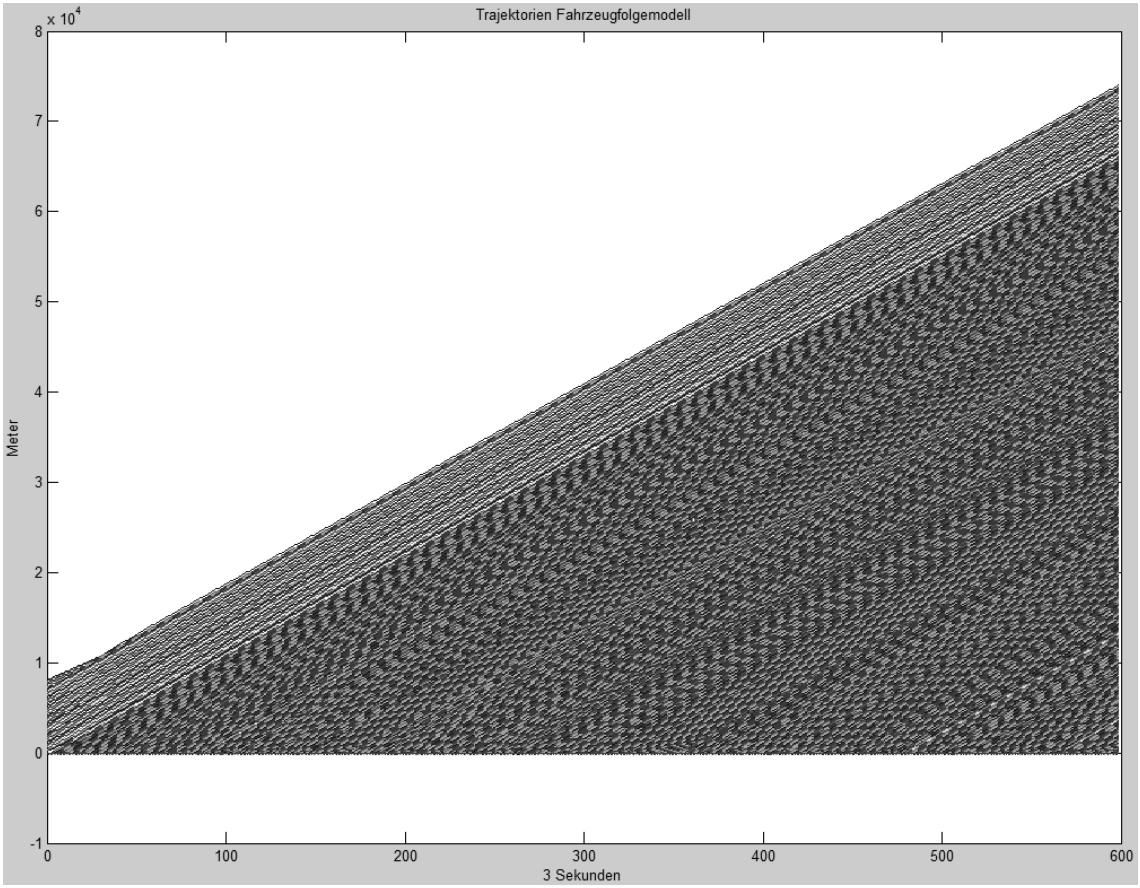

Abbildung 5.1: Trajektorien Fahrzeugfolgemodell

Im Trajektorienausschnitt in Abbildung 5.2, Seite 94 lässt sich sehr gut das Charakteristische des Fahrzeugfolgemodells erkennen. Die FahrerFahrzeuge passen sich exakt ihrem Vordermann an, was sich im parallelen Verlauf der Trajektorien zeigt. Deutlich erkennbar sind auch die teils extrem geringen Abstände der FahrerFahrzeuge zueinander. Grundsätzlich verlaufen die Trajektorien sehr glatt und unterliegen nur minimalen Schwankungen.

Abbildung 5.2: Ausschnitt Trajektorien Fahrzeugfolgemodell

5.2.3 Wiedemann-Modell

Der Verlauf der Trajektorien über den gesamten Messzeitraum in Abbildung 5.3, Seite 95 zeigt, dass sich der Verkehr in Pulks bewegt, zwischen denen sich größere Lücken befinden. Diese Pulkbildung entsteht durch die unterschiedlichen Wunschgeschwindigkeiten der FahrerFahrzeuge. Ein Pulk bildet sich dann, wenn FahrerFahrzeuge mit einer höheren Wunschgeschwindigkeit auf eines mit niedrigerer auffahren und diesem schließlich folgen. Ein FahrerFahrzeug, mit einer ähnlichen Wunschgeschwindigkeit, wie die des vor ihm den Pulk anführenden FahrerFahrzeuges, schließt nicht zu diesem auf. Es lässt eine Lücke entstehen und zieht seinen eigenen Pulk hinter sich her. Zudem ist eine Verdichtung des Verkehrs mit fortlaufender Zeit ersichtlich.

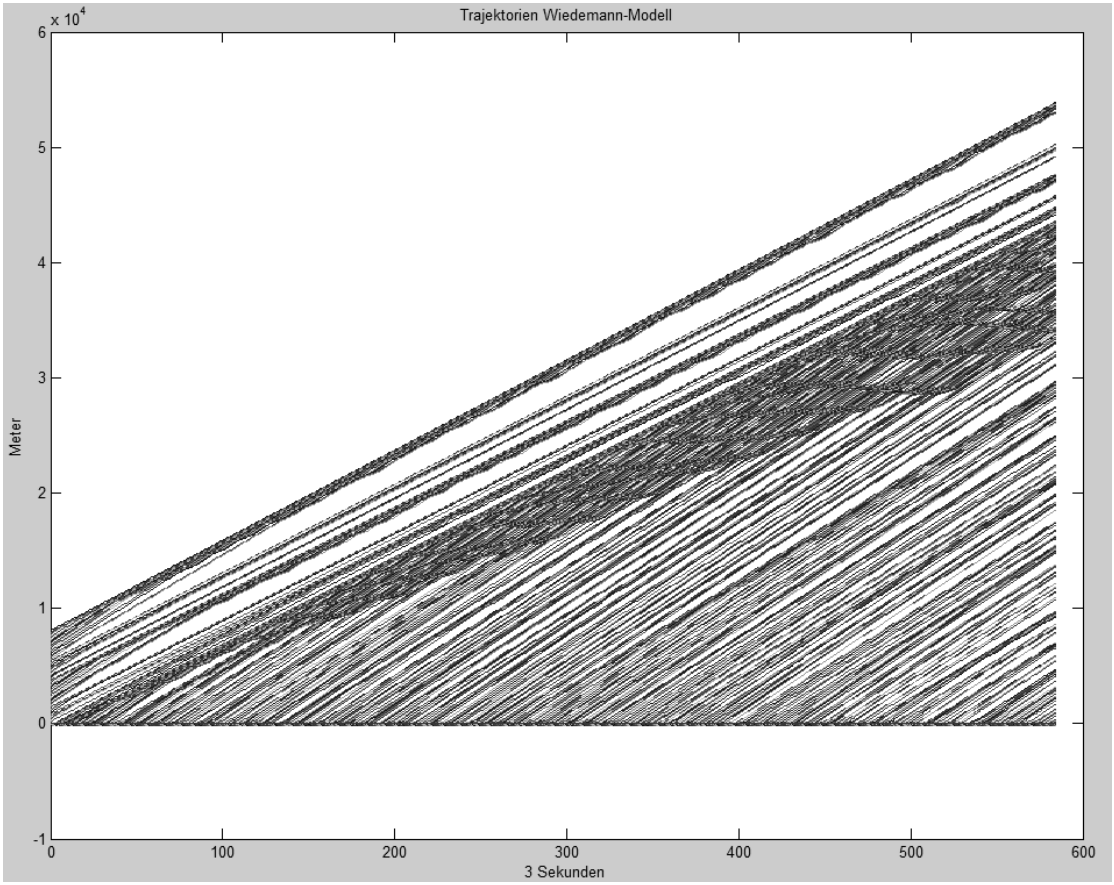

Abbildung 5.3: Trajektorien Wiedemann-Modell

Die Verkehrsverdichtung ist in Abbildung 5.4, Seite 96 gut zu erkennen. Es ist deutlich zu sehen, dass die einzelnen Pulks zueinander aufschließen und sich eine große Kolonne bildet. Das Aufschließen eines Pulks geschieht immer dann, wenn sich durch die Kolonne eine Verzögerungsgswelle bewegt, wodurch deren Geschwindigkeit kurzfristig unter der des aufschließenden Pulkes sinkt. Die Verzögerungsgswellen entstehen, wenn ein FahrerFahrzeug seinen gewünschten Sicherheitsabstand unterschreitet. Dies zwingt ihn zu einer Verzögerung, um den Abstand wieder zu vergrößern, was daraufhin zum Unterschreiten des Sicherheitsabstandes seines Hintermanns führt und diesen wiederum zu einer Verzögerung nötigt. Das Fortsetzten dieses Ablaufes drückt sich durch diese Wellen aus. Diese Verzögerungswellen sind auch in den Pulks vor der Kolonne erkennbar. Insgesamt lässt sich bei näherer Betrachtung der einzelnen Trajektorien sagen, dass sie inhomogen verlaufen und schwanken.

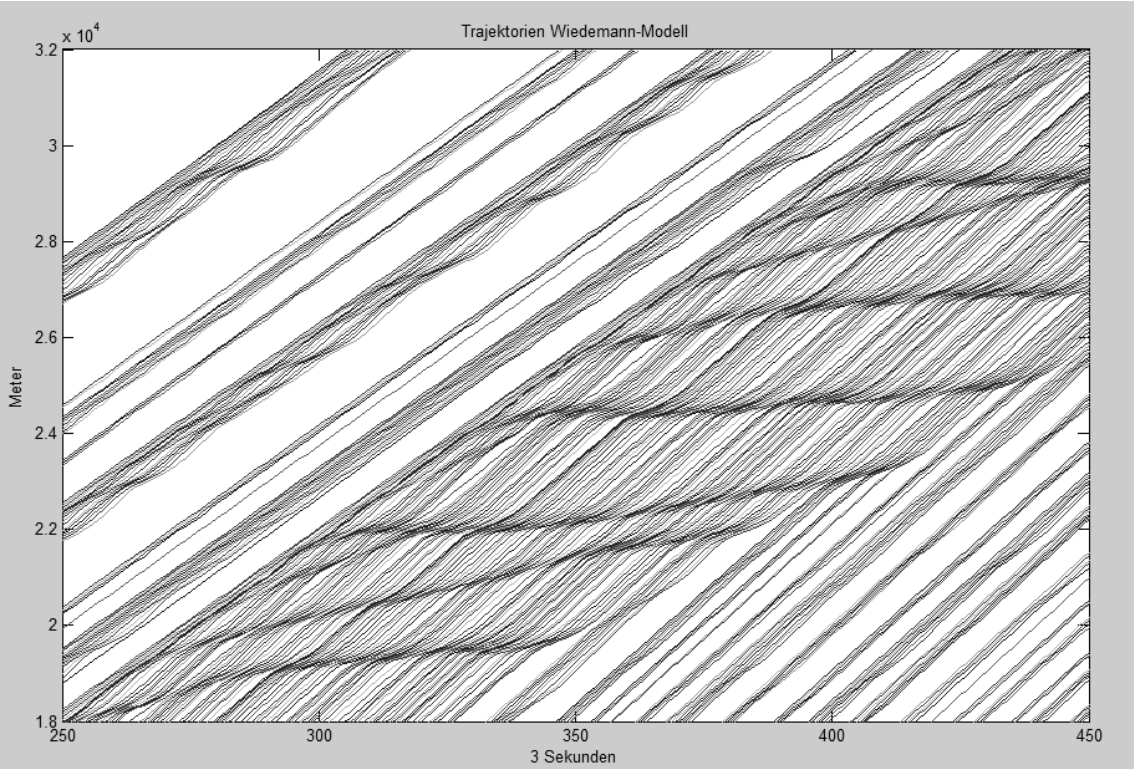

Abbildung 5.4: Ausschnitt Trajektorien Wiedemann-Modell

5.2.4 Erweitertes Wiedemann-Modell

Bei Betrachtung des gesamten Trajektorienverlaufs in Abbildung 5.5, Seite 97 geht hervor, dass sich die FahrerFahrzeuge in Pulks fortbewegen, zwischen denen sich große Lücken befinden. Der Grund dafür wurde bereits im vorangegangen Abschnitt beschrieben. Es ist weiterhin deutlich erkennbar, dass es zu keinem Aufschließen der Pulks kommt und keine Kolonnenbildung stattfindet.

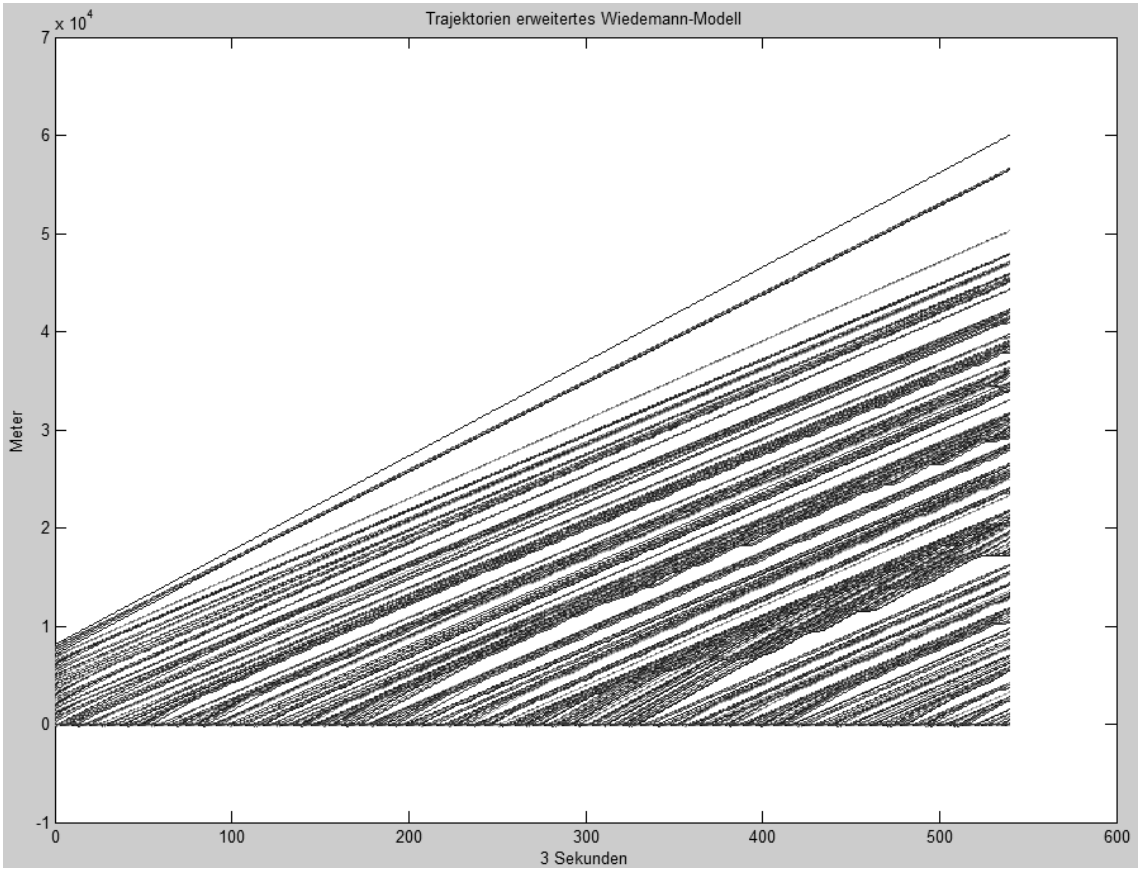

Abbildung 5.5: Trajektorien erweitertes Wiedemann-Modell

Abbildung 5.6, Seite 98 zeigt, dass sich in dem großen Pulk Stop-and-Go-Wellen gebildet haben. Es ist deutlich zu erkennen, wie sich diese Wellen aus Verzögerungswellen heraus entwickeln. Die Stop-and-Go-Wellen entstehen, wenn während einer Verzögerungswelle ein FahrerFahrzeug anfängt zu trödeln (schwarze Ellipse). Nach Beginn des Trödelns verringert sich zunächst die momentane Geschwindigkeit dieses FahrerFahrzeuges, wodurch es zu einem Abreißen vom Vordermann kommt. Wenn die Trödelphase beendet ist, strebt es wieder seine Wunschgeschwindigkeit an und schließt die entstandene Lücke. Damit eine Stop-and-Go-Welle sich aber überhaupt aus einer Verzögerungswelle bilden kann, muss der Pulk hinreichend groß sein.

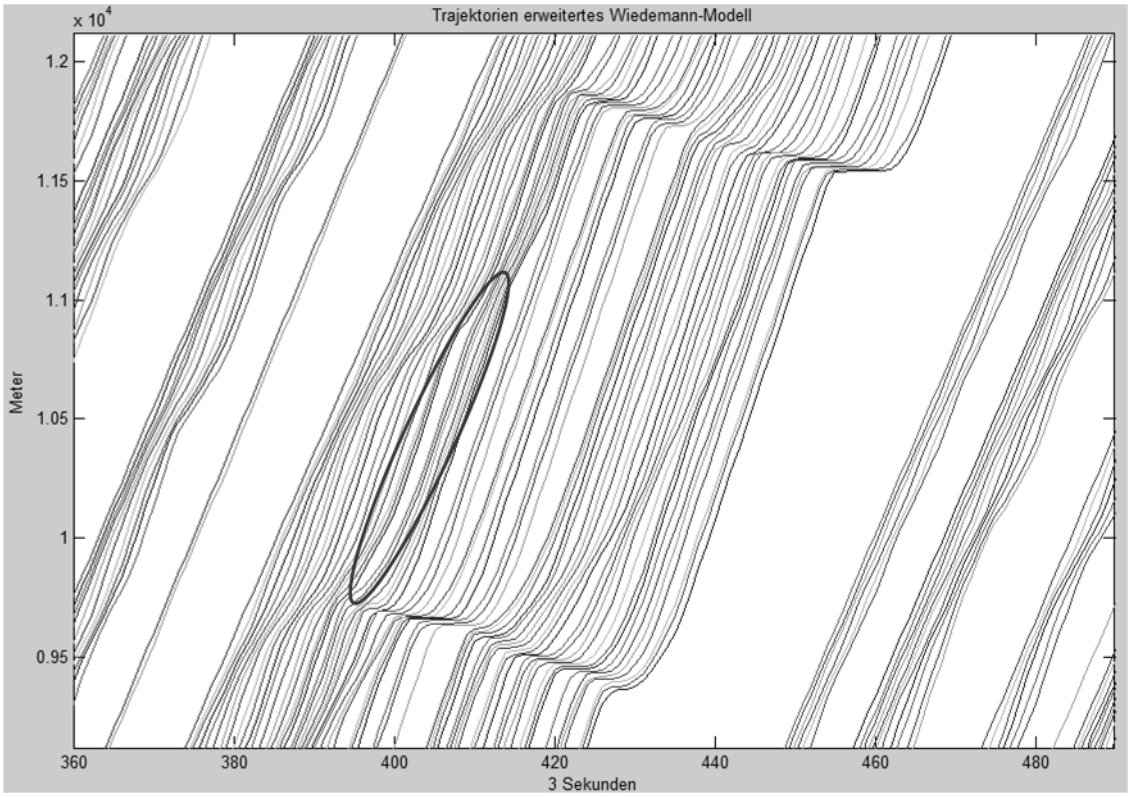

Abbildung 5.6: Ausschnitt Trajektorien erweitertes Wiedemann-Modell

5.2.5 Vergleich und Bewertung

In diesem Abschnitt sollen die implementierten Verkehrsmodelle anhand der Aussagen aus den drei vorangegangen Abschnitten verglichen und bewertet werden.

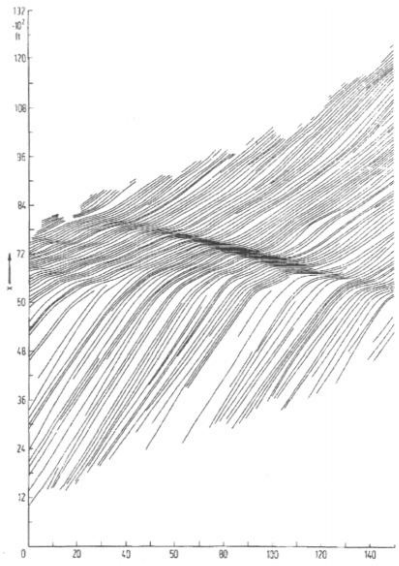

Abb. 11.1. Trajektorien von Fahrzeugen, wie sie von Treiterer *et al.* [444] aus Luftaufnahmen gewonnen wurden. Plötzlich endende oder beginnende Linien sind durch Spurwechsel bedingt. Die Zeit t (in s) ist nach rechts aufgetragen, der Ort x (in 100 ft) nach oben. Daher entspricht die Steigung der Linien gerade der Geschwindigkeit der Fahrzeuge (in 100 ft/s). Deutlich erkennbar ist die Ausbildung einer Stop-and-Go-Welle, die sich entgegen der Fahrtrichtung ausbreitet (aus [268]).

Abbildung 5.7: Reale Trajektorien (Entnommen aus [Helb97] Seite 78)

Anhand eines Vergleichs der Trajektorien realer Fahrzeuge in Abbildung 5.7, Seite 98 mit denen der untersuchten Verkehrsmodelle bestätigt sich, dass sowohl das ursprüngliche Wiedemann-Modell als auch das erweiterte Wiedemann-Modell eine hohe Realitätsnähe besitzen. Vom Fahrzeugfolgemodell lässt sich hingegen nur sagen, dass es ein reales Verkehrsgeschehen nicht nachbildet. Es erzeugt lediglich einen Strom hintereinander fahrender FahrerFahrzeuge, deren Abstände zueinander konstant und vollkommen unabhängig von der gefahrenen Geschwindigkeit sind. Es ist festzuhalten, dass die Versuche, Schwankungen der Geschwindigkeiten durch eine verzögerte Reaktion der FahrerFahrzeuge auszulösen, nicht sonderlich erfolgreich waren. Allenfalls sind minimale Schwankungen der Trajektorien erkennbar, die jedoch kaum mit denen der Wiedemann-Modelle zu vergleichen sind. Die einzige Ausnahme ist, dass die Trajektorien von einzelnen unbeeinflussten und frei fahrenden FahrerFahrzeugen in den Wiedemann-Modellen ebenfalls nur sehr geringe Schwankungen aufweisen. Je mehr FahrerFahrzeuge aber in einem Verband fahren und sich gegenseitig beeinflussen, desto unregelmäßiger werden die Trajektorien. Im ursprünglichen Wiedemann-Modell ist zudem wahrnehmbar, dass die Verzögerungswellen in der großen Kolonne steiler sind als in den kleineren Pulks. Die FahrerFahrzeuge werden also stärker abgebremst.

Im Vergleich der beiden Wiedemann-Modelle ist festzustellen, dass sich beim erweiterten Wiedmann-Modell keine große Kolonne bildet. Hervorgerufen wird dieser Effekt durch das Trödeln einzelner FahrerFahrzeuge, was das Aufschließen von Pulks während der Verlangsamungen verhindert. Trotz der großen Ähnlichkeiten der beiden Wiedemann-Modelle unterscheiden sie sich doch in einem Punkt deutlich: Das erweiterte Wiedemann-Modell erzeugt die empirisch belegten Stop-and-Go-Wellen „aus dem Nichts“, während sie beim ursprünglichen Wiedemann-Modell nicht auftreten. Damit sich eine Stop-and-Go-Welle bilden kann, müssen drei Bedingungen zutreffen:

1. Muss der FahrerFahrzeug-Pulk hinreichend groß sein.
2. Muss bereits eine Verzögerungswelle ablaufen.
3. Muss in dieser Situation ein FahrerFahrzeug anfangen zu trödeln.

Beim ursprünglichen Wiedemann-Modell bildet sich auch in der sehr großen Kolonne keine Stop-and-Go-Welle.

Zusammenfassend ist festzustellen: Das erweiterte Wiedemann-Modell weist die größte Realitätsnähe auf und sollte deswegen bevorzugt verwendet werden. Einen etwas geringeren Realitätsgrad hat das ursprüngliche Wiedemann-Modell, das keine Stop-and-Go-Wellen erzeugt. Trotzdem kann sein Einsatz empfohlen werden, da sich diese Wellen nur bei hohen Verkehrsstärken bilden können. Zudem kann das Auftreten von Stop-and-Go-Wellen nicht immer erwünscht sein. Nicht empfehlenswert für den Einsatz ist hingegen das Fahrzeugfolgemodell, da der dabei erzeugte Verkehrsstrom kaum etwas mit einer realen Verkehrssituation gemein hat.

5.3 Simulierter Verkehr mit externem Fahrzeug

5.3.1 Vorbemerkungen

Da der simulierte Verkehr des Verkehrssimulationsmoduls hauptsächlich eingesetzt werden soll, um das Realitätsgefühl bei menschlichen Fahrern zu steigern, wurden Untersuchungen mit sieben Probanden durchgeführt. Sie fuhren dabei an einem Desktop-Simulator dreimal die gleiche Strecke ab, wobei der Verkehr jeweils mit einem anderen Verkehrsmodell simuliert wurde. Nach jeder Fahrt haben sie auf einem Auswertungsblatt die Fragen bezüglich des Realitätsgrades des Verkehrs auf ihrer eigenen Fahrspur und auf der Gegenfahrspur beantwortet. Dabei sollten sie zunächst einen Punktewert auf einer Skala von 1 bis 10 vergeben (1 – gar nicht realitätsnah; 10 – sehr realitätsnah) und anschließend in einem Freitextfeld ihre Eindrücke aufschreiben. Bevor die Probanden mit den Testfahrten begannen, konnte sich jeder von Ihnen mit dem Simulator vertraut machen und sich eine gewisse Zeit einfahren. Sobald die Probanden meinten, sie hätten sich eingewöhnt, starteten die Testfahrten. Dadurch sollte erreicht werden, dass sie sich ab der ersten Testfahrt auf das Fahren im Verkehr konzentrieren können und nicht durch mangelnde Handhabungsfähigkeiten abgelenkt sind.
Die Versuchsbedingungen, die bei allen Testfahrten galten, sind in Tabelle 5.2, Seite 101 aufgeführt. Es wurde eine relativ niedrige Verkehrsstärke gewählt, damit sich Überholmöglichkeiten ergaben.
Die ausgefüllten Fragebögen sind im Anhang C dieser Masterarbeit beigefügt. Die eine weibliche Probandin und die sechs männlichen Probanden besitzen alle einen Führerschein der Klasse B und sind zwischen 22 und 27 Jahre alt. Bei jeder Testfahrt wurden außerdem verschiedene Werte, wie die Trajektorien aller FahrerFahrzeuge auf der eigenen Fahrspur und die Geschwindigkeit des Probandenfahrzeuges, aufgezeichnet.
Es sei noch angemerkt, dass aufgrund der geringen Stichprobenanzahl die Ergebnisse nicht als allgemeingültig bewertet werden können. Deshalb sind die in den folgenden Abschnitten getroffenen Aussagen unter Vorbehalt aufzufassen. Es lassen sich aber trotzdem gewisse Tendenzen erkennen.
Alle Rohdaten und Auswertungsdateien liegen dieser Masterarbeit auf der CD im Verzeichnis *.\Tests\ProbandentestsVerkehr* bei.

Tabelle 5.2: Versuchsbedingungen der Testläufe

Parameter	Parameterwert
Streckenlänge	7852 m
Fahrspuren	1 in Fahrtrichtung 1 in Gegenfahrtrichtung
Verkehrsstärke	500 Fz/h
Fahrspuren	2
SV-Anteil	20 %
SV-Geschwindigkeitsverteilung	Erwartungswert $\mu = 90\ km/h$ Standardabweichung $\sigma = 1.67\ km/h$
PKW-Geschwindigkeitsverteilung	Erwartungswert $\mu = 110\ km/h$ Standardabweichung $\sigma = 6\ km/h$
Hardware	Windows XP Professional; Intel Core 2 Quad Q9550 2.83 GHz; 3.25 GB RAM
Prozessorkerne	4

5.3.2 Fahrzeugfolgemodell

Die Zahlen der Tabelle 5.3, Seite 102 zeigen, dass die Probanden zu sehr unterschiedlichen Bewertungen des Realitätsgrades des Verkehrs sowohl auf der eigenen als auch auf der Gegenfahrspur kommen. Besonders groß sind die Diskrepanzen bei der Einschätzung auf der eigenen Fahrspur, was sich auch durch eine hohe Standardabweichung ausdrückt. Weiterhin ist festzuhalten, dass der Verkehr auf der eigenen Fahrspur im Durchschnitt realer eingeschätzt wird als auf der Gegenfahrspur. Zusammenfassend wird der erzeugte Verkehr durch das Fahrzeugfolgemodell als nicht besonders realitätsnah bewertet.

Auch bei den mittleren Geschwindigkeiten treten relativ große Schwankungen auf. Sie sind aber durch die Geschwindigkeitsverteilung und der Eigenschaft des Fahrzeugfolgemodells zu erklären, wonach sich alle FahrerFahrzeuge der Geschwindigkeit des Ersten FahrerFahrzeuges anpassen. Zu den beiden Extremwerten der Probanden 6 und 7 ist festzustellen, dass sie in der 3σ-Umgebung der angegebenen Geschwindigkeitsverteilung liegen. Der Mittelwert der mittleren Geschwindigkeiten liegt dann auch mit 108.22 km/h sehr nah am vorgegebenen Erwartungswert von 110 km/h. Einzig die Standardabweichung ist mit 9.07 km/h deutlich größer als die vorgegebenen 6 km/h. Diese Abweichung kann aber durch die geringe Stichprobengröße erklärt werden.

Tabelle 5.3: Auswertungstabelle beim Fahrzeugfolgemodell

Fahrzeugfolgemodell				
Proband	Realitätsgrad Gegenfahrspur	Realitätsgrad Eigene Spur	Realitätsgrad Mittelwert	Mittlere Geschwindigkeit (m/s)
1	6	3	4.5	27.45182462
2	5	x	-	29.13995573
3	2	1	1.5	31.16287002
4	3	8	5.5	31.20197706
5	2	6	4	30.30508566
6	5	8	6.5	34.28501495
7	6	7	6.5	26.89048436
Mittelwert	4.142857143	5.5	4.75	30.06245891
Standardabweichung	1.772810521	2.880972058	1.89076704	2.52078359

In den Freitext-Antworten bezüglich des erzeugten Gegenverkehrs trat maßgeblich ein Kritikpunkt hervor:

- Die Abstände zwischen den Fahrzeugen waren gleichmäßig groß, wodurch kein Überholen möglich war.

Dies ist das typische Charakteristikum des Fahrzeugfolgemodells, das bereits in Abschnitt 5.2.2 erwähnt wurde. Weitere Kritikpunkte waren, dass alle Fahrzeuge (auch LKW) gleichmäßig schnell fuhren und dass der Gegenverkehr nicht reagierte, um Zusammenstöße zu verhindern. Die gleichmäßige Geschwindigkeit ist ein weiteres typisches Merkmal des Fahrzeugfolgemodells. Da sich das Modell nur auf das Hinterherfahren beschränkt und keine Spurwechselmodelle implementiert sind, war ein Reagieren des Gegenverkehrs nicht möglich. Zudem kann auch nicht festgestellt werden, wann das externe Fahrzeug seine Spur verlässt. Die dafür benötigte Information über die Querabweichung von der Mittellinie der Straße wird von CarMaker nicht aufgezeichnet und kann deswegen nicht vom Kommunikationsmodul an das Visualisierungsmodul gesendet werden.

Wie nach den Bewertungen in der Tabelle 5.3 bezüglich des Realitätsgrades der eigenen Spur zu erwarten war, unterscheiden sich auch die Freitext-Antworten deutlich voneinander. Einige Probanden befanden den Verkehr als zumindest *„relativ realistisch“*. Häufig wurden auch wieder die unrealistischen Abstände und Geschwindigkeiten kritisiert. Allerdings empfand ein Proband den vor ihm schleichenden *„LKW sehr realitätsnah“*. Ein Proband schrieb, dass ihm Überholvorgänge fehlten. Dazu ist anzumerken, dass keine Spurwechselmodelle integriert sind.

Ein ernstzunehmender Kritikpunkt eines einzelnen Probenden war, dass ein Fahrzeug durch ihn durchgefahren sei. Werden die Umstände betrachtet, unter denen es dazu kam, kann dieser Kritikpunkt aber entkräftet beziehungsweise durch die Eigenschaften des Fahrzeugfolgemodells erklärt werden. Zu dem Hindurchfahren kam es, als der Proband ein Fahrzeug

überholt hatte und kurz nach dem Einordnen stark bremste. Das überholte Fahrzeug hatte seinen Abstand auf das Fahrzeug des Probanden noch nicht vergrößert. Somit betrug dieser nur wenige Meter. Dass unter diesem Umstand und der hohen Geschwindigkeit von über 100 km/h ein plötzliches Abbremsen des Probanden zu einem „Unfall“ führte, ist sogar eher als realistisch zu bewerten.

5.3.3 Wiedemann-Modell

Die Wertungen in Tabelle 5.4 zeigen, dass insbesondere der Verkehr auf der eigenen Fahrspur sehr unterschiedlich eingeschätzt wurde. Dies drückt sich wieder in einer deutlich größeren Standardabweichung als bei der Gegenfahrspur aus. Im Durchschnitt sind der Realitätsgrad des Verkehrs auf der Gegenfahrspur und auf der eigenen Fahrspur als nahezu gleich bewertet worden. Es ist auch zu erkennen, dass die Probanden den Verkehr mit durchschnittlich rund 7.5 als relativ realitätsnah empfanden. Die mittleren Geschwindigkeiten sind bis auf Proband 6 bei allen Probanden nahezu gleich.

Tabelle 5.4: Auswertungstabelle beim Wiedemann-Modell

Wiedemann-Modell				
Proband	Realitätsgrad Gegenfahrspur	Realitätsgrad Eigene Spur	Realitätsgrad Mittelwert	Mittlere Geschwindigkeit (m/s)
1	6	6	6	29.43855043
2	10	10	10	28.78254246
3	7	3	5	29.32934926
4	7	9	8	28.52283302
5	9	8	8.5	28.72252856
6	7	8	7.5	33.01854844
7	7	7	7	27.72086794
Mittelwert	7.571428571	7.285714286	7.428571429	29.36217430
Standardabweichung	1.397276262	2.288688541	1.643892014	1.70877552

Bei den Freitext-Antworten der Probanden bezüglich des Gegenverkehrs wurde fast immer betont, dass dieser realitätsnah sei. Ein Proband bezeichnete ihn sogar als *„Perfekt“*. Häufig wurde geäußert, dass hier insbesondere die Abstände der Fahrzeuge realistisch erscheinen und auch Lücken zum Überholen da sind. Ein Proband merkte allerdings an, dass der Gegenverkehr dicht gewesen sei und es kaum Überholmöglichkeiten gab. Er hatte in diesem Fall einfach „Pech“, dass der Verkehr zufällig so erzeugt wurde. Ein Proband schrieb, dass hinter einem LKW zu wenige PKW hinterher fahren. Das kann einerseits am relativ hohen SV-Anteil von 20 % liegen, andererseits können durch das fehlende Überholen der PKW längere Kolonnen eben nur dann entstehen, wenn jene zufällig so erzeugt werden. Ein Proband

bemerkte auch hier, dass der Gegenverkehr in kritischen Überholsituationen nicht reagiert. Die Gründe dafür wurden bereits im vorherigen Abschnitt genannt.
Auch die Freitext-Antworten zum Verkehr auf der eigenen Fahrspur drücken größtenteils die relativ hoch empfundene Realitätsnähe aus. Es lassen sich drei Hauptkritikpunkte aus den Antworten herauslesen.

1. Es findet kein Überholen durch die simulierten Fahrzeuge statt.

Als Grund wurden die fehlenden Spurwechselmodelle bereits genannt.

2. Das vorausfahrende Fahrzeug lässt sich nicht durch Drängeln beeinflussen.

Dies ist darin begründet, dass im Wiedemann-Modell eine Beeinflussung durch den rückwärtigen Verkehr als Sonderfall ausgeschlossen und nicht mit betrachtet wird.

3. LKW sind bergauf zu schnell.

Weil es im Verkehrssimulationsmodul keine geometrische Abbildung der Straßen gibt, kann auch keine Beeinflussung auf die Beschleunigung der FahrerFahrzeuge durch die geometrischen Eigenschaften der Straße stattfinden.
Eine weitere Anmerkung von Probanden war, dass sie die Geschwindigkeiten kaum einschätzen konnten. Dies liegt in dem zu geringen Immersionseindruck des Simulators begründet. Damit lässt sich auch zum Teil die Aussage eines Probanden erklären, er hätte „*Stellenweise unerklärliche Beschleunigung der Gruppe vor dem eigenen Fahrzeug*“ wahrgenommen. Möglich ist aber auch, dass die FahrerFahrzeuge nach einer Verzögerungswelle wieder beschleunigten oder sich diese Aussage auf die Nichtberücksichtigung der Geometrie der Straße auf das Beschleunigungsverhalten der FahrerFahrzeuge bezieht.
All diese kritisch angemerkten Punkte scheinen den Realitätseindruck der Probanden jedoch nur in eher geringem Maße getrübt zu haben, was sich in der durchschnittlichen Realitätsbewertung von rund 7.5 ausdrückt.

5.3.4 Erweitertes Wiedemann-Modell

Bei der Auswertung der Tabelle 5.5, Seite 105 fällt auf, dass bis auf Proband 1 alle anderen den Gegenverkehr nahezu gleich als sehr realistisch bewerten. Dies drückt sich in der hohen durchschnittlichen Bewertung von rund 8.7 aus. Deutlichere Bewertungsunterschiede treten wieder bei der Einschätzung der eigenen Fahrspur auf. Hier wird der Realitätsgrad mit circa 7.4 geringer als auf der Gegenfahrspur angesehen. Die Bewertung des gesamten Verkehrs mit annähernd 8.1 fällt insgesamt jedoch sehr gut aus. Deutliche Unterschiede zwischen den Probanden treten bei ihrer gefahrenen Durchschnittsgeschwindigkeit auf, was in der hohen Standardabweichung zum Ausdruck kommt. Dies kann dadurch verursacht sein, dass die Probanden mit einer vergleichsweise geringen mittleren Geschwindigkeit in einem Pulk mit einem trödelnden FahrerFahrzeuge gefahren sind.

Tabelle 5.5: Auswertungstabelle beim erweiterten Wiedemann-Modell

Erweitertes Wiedemann-Modell				
Proband	Realitätsgrad Gegenfahrspur	Realitätsgrad Eigene Spur	Realitätsgrad Mittelwert	Mittlere Geschwindigkeit (m/s)
1	6	6	6	31.14532263
2	10	10	10	27.39609911
3	9	6	7.5	34.15818685
4	9	5	7	27.43125297
5	9	8	8.5	34.00083829
6	9	8	8.5	34.85096806
7	9	9	9	29.45851986
Mittelwert	8.714285714	7.428571429	8.07142857	31.20588397
Standardabweichung	1.253566341	1.812653934	1.33630621	3.20504157

Außer bei Proband 1 wird in den Freitext-Antworten der Verkehr als sehr realitätsnah bewertet. Ein Proband bezeichnete ihn sogar als *„Perfekt“*. Die Probanden merkten häufig an, dass sie die Abstände zwischen den Fahrzeugen sowie deren Geschwindigkeiten als sehr realistisch empfanden. Es wurde nur von einem Probanden als ein Kritikpunkt geäußert, dass der Gegenverkehr in kritischen Überholsituationen nicht reagiert. Die Gründe dafür sind im Abschnitt 5.3.2 erläutert.

In den Freitext-Antworten der Probanden bezüglich des Realitätsgrades auf der eigenen Fahrspur wurde häufig geschrieben, dass sie keinen Unterschied zum ursprünglichen Wiedemann-Modell feststellen konnten. Von daher gelten die gleichen Kritikpunkte wie im vorangegangenen Abschnitt. Ein Proband schrieb weiterhin, dass ihm die *„Autos manchmal ziemlich langsam“* vorkamen. Das kann sowohl an der schlechten Wahrnehmbarkeit von Geschwindigkeiten am Simulator liegen oder daran, dass er sich in einem Pulk mit einem trödelnden Fahrzeug befand. Ein weiterer Proband *„Hatte den Eindruck*[,] *das*[s] *auf meiner Spur viel weniger Fahrzeuge fuhren“*. Die Ursache dafür war, dass der Proband an einer Stelle im Verkehrsstrom fuhr, an der sich zufällig weniger Fahrzeuge befanden. Ein Proband führte aus, dass die „*Beschleunigung der anderen Verkehrsteilnehmer oft zu hoch ist*[,] *da bei eigenen Bremsvorgängen oft Probleme bestehen*[,] *wieder aufzuschließen“*. Das ist deshalb so, weil das durch CarMaker simulierte Fahrzeug des Probanden die Fahrzeugdynamik sehr viel realistischer abgebildet hat, während dies bei den durch das Verkehrssimulationsmodul simulierten Fahrzeugen aus Vereinfachungsgründen nicht der Fall ist.

5.3.5 Vergleich und Bewertung

Zusammengefasst ergeben die Bewertungen der drei implementierten Verkehrsmodelle aus den drei vorangegangen Abschnitt eindeutig, dass das erweiterte Wiedemann-Modell am besten abschneidet und als am realistischsten angesehen wird. Mit geringem Abstand folgt dahinter das ursprüngliche Wiedemann-Modell. Als eher unrealistisch erscheint den Probanden das Fahrzeugfolgemodell. Das drücken auch die Gesamtbewertungen in den Fragebögen der Probanden aus. Bei allen Modellen wurde kritisiert, dass der Gegenverkehr bei riskanten Überholmanövern nicht reagiert und dass die Fahrzeuge nicht überholen. Die deutlichen Bewertungsunterschiede zwischen dem Fahrzeugfolgemodell und den Wiedemann-Modellen ergeben sich hauptsächlich aus der Realitätseinschätzung der Abstände zwischen den Fahrzeugen und ihren Geschwindigkeiten. Beides wird beim Fahrzeugfolgemodell als unrealistisch angesehen. Die Bevorzugung des erweiterten Wiedemann-Modells gegenüber dem ursprünglichen ergibt sich vorwiegend aus der deutlich besseren Bewertung des Gegenverkehrs. Auf der eigenen Fahrspur wird er fast gleich bewertet.
Allen Probanden fiel es schwer, den Realitätsgrad auf der eigenen Fahrspur zu bewerten, was sich in der durchweg größeren Standardabweichung gegenüber der Gegenfahrspur ausdrückt. Ein vorausfahrendes Fahrzeug scheint bei den Probanden 5 bis 7 maßgebend für eine realistische Einschätzung des Verkehrs auf der eigenen Fahrspur zu sein.

5.4 Fazit

Beim Vergleich der Ergebnisse aus Abschnitt 5.2 mit denen aus Abschnitt 5.3 bestätigen sich die Bewertungen der drei implementierten Verkehrsmodelle. Sowohl bei der Betrachtung der Trajektorien der FahrerFahrzeuge als auch nach Einschätzungen der Probanden geht das erweiterte Wiedemann-Modell als bestes dieser drei Modelle hervor. Auch wurde in beiden Abschnitten das ursprüngliche Wiedemann-Modell knapp dahinter platziert und das Fahrzeugfolgemodell als unrealistisch angesehen. Beim Einsatz des Verkehrssimulationsmoduls sollte also das erweiterte Wiedemann-Modell verwendet werden.

6 Diskussion

6.1 Zielerfüllung

Die in Abschnitt 1.3 gestellten Fragen sollten durch diese Masterarbeit beantwortet werden. In diesem Abschnitt wird dabei im Wesentlichen auf die Stellen in dieser Masterarbeit verwiesen, in denen die Antworten auf diese Fragen zu finden sind.

1. ***Welche Verkehrsmodelle gibt es? Wie sind sie zu klassifizieren? Welche eignen sich für diesen Anwendungszweck?***

In den Abschnitten 2.3.2, 2.3.3.1 und 2.3.5 werden einige Verkehrsmodelle genannt und in den Abschnitten 2.3.3.2 bis 2.3.3.4 drei Modelle beschrieben. Eingeteilt werden können sie in makroskopische, mikroskopische und mesoskopische Verkehrsmodelle. Diese Klassifizierung geht auch aus der Struktur des Abschnittes 2.3 hervor. Für die Realisierung des Verkehrssimulationsmoduls eignen sich nur mikroskopische Modelle, da nur diese das einzelne FahrerFahrzeug betrachten; siehe dazu auch Abschnitt 2.3.6.

2. ***Welche Verkehrsmodelle sollen implementiert werden?***

Es wurden die in den Abschnitten 2.3.3.2 bis 2.3.3.4 beschriebenen Modelle implementiert. Deren Modellierung und Implementierung werden in den Abschnitten 3.5.4 und 4.4.3 erläutert.

3. ***Welcher Funktionsumfang kann in der vorgegebenen Bearbeitungszeit der Masterarbeit umgesetzt werden?***

Es konnten der Funktionsumfang umgesetzt werden, der durch die Anforderungen im Abschnitt 3.2 definiert ist.

4. ***Wie ist das Verkehrssimulationsmodul zu modellieren? Wie kann dabei die möglichst leichte Erweiterbarkeit hinsichtlich zusätzlicher Verkehrsmodelle gewährleistet werden? Wie kann eine möglichst hohe Plattformunabhängigkeit erreicht werden?***

Die Modellierung des Verkehrssimulationsmoduls ist im Kapitel 3 ausführlich beschrieben. Die Gewährleistung der leichten Erweiterbarkeit geht aus den Beschreibungen der Modellierung der Modellkomponenten im Abschnitt 3.5 hervor. Im Anhang B.1 wird an einem Beispiel erläutert, welche Schritte für die Integration eines weiteren Verkehrsmodells durchgeführt werden müssten. Durch den Einsatz der plattformunabhängigen Bibliotheken von boost konnten insbesondere die Parallelisierung und die Netzwerkkommunikation, beschrieben in den Abschnitten 4.4.4 und 4.4.5, weitgehend plattformunabhängig gestaltet werden.

5. ***Wie gliedert sich das Verkehrssimulationsmodul in das bestehende Verteilte Softwaresystem ein? Welche Daten müssen zwischen den verschiedenen Modulen ausgetauscht werden?***

Die Einordnung des Verkehrssimulationsmoduls in das Softwaresystem „RoadMaker“ ist im Abschnitt 3.3 erläutert. Die Beschreibung der ausgetauschten Daten erfolgt in den Abschnitten 3.5.6 und 4.4.5.

6. ***Wie kann die Berücksichtigung des externen CarMaker-Fahrzeuges bei der Simulation des Verkehrs erfolgen? Wie kann das Modul sinnvoll parallelisiert werden? Welches Zeitverhalten hat es?***

Die Integration des externen CarMaker-Fahrzeuges ist in den Abschnitten 3.5 und 4.4 beschrieben. Die Parallelisierung erfolgt nach Fahrspuren und wird durch die Abschnitte 3.5.5 und 4.4.4 erläutert. Im Abschnitt 4.8 erfolgt die Beschreibung des Zeitverhaltens der Simulation.

7. ***Welche Parametrisierungsmöglichkeiten können sinnvollerweise angeboten werden, um den flexiblen Einsatz des Verkehrssimulationsmoduls zu gewährleisten?***

Die vielfältigen Parametrisierungsmöglichkeiten sind im Abschnitt 4.6 dargelegt.

8. ***Wie ist die Güte des erzeugten Verkehrs der verschieden Modelle? Wie bewerten Menschen den Realitätsgrad des simulierten Verkehrs, wenn sie in ihm fahren?***

Die Antworten auf diese Fragen werden im Kapitel 5 gegeben. Dabei erfolgt die Betrachtung der Güte anhand der Trajektorien der FahrerFahrzeuge im Abschnitt 5.2. Das Empfinden des Realitätsgrades der realisierten Verkehrsmodelle durch Menschen ist im Abschnitt 5.3 beschrieben.

6.2 Ergebnisse und Ausblick

Als Ergebnis dieser Masterarbeit steht ein Softwaremodul, das auf Basis von mikroskopischen Verkehrsmodellen einen Verkehrsstrom simuliert. In ihm kann sich ein Mensch mit einem externen Fahrzeug bewegen. Das Softwaremodul stellt zudem eine Komponente eines Verteilten Systems dar. Es wurden Verkehrsmodelle vorgestellt und klassifiziert und drei mikroskopische Verkehrsmodelle umfassend erläutert. Ferner erfolgte eine ausführliche Beschreibung dazu, wie diese Verkehrsmodelle modelliert, algorithmiert und schließlich parallelisiert implementiert werden können.

Die Test mit den Probanden und die Betrachtungen der Trajektorien der simulierten Fahrer-Fahrzeuge zeigten eines: Das Verkehrssimulationsmodul erzeugt auf Basis des erweiterten Wiedemann-Modells oder des ursprünglichen Wiedemann-Modells einen durchaus realistischen Verkehrsfluss auf Straßen mit zwei unterschiedlichen Richtungsfahrbahnen. Damit ist die Hauptanforderung an das Modul erfüllt. Es zeigte sich auch, dass das Fahrzeugfolgemodell ungeeignet für den Einsatz ist.

Trotz des erreichten Standes sieht der Autor noch großes Entwicklungspotential für viele Funktionen. Zu allererst sollte das Modul um Spurwechselverhalten erweitert werden, damit

auch realistischer Verkehr bei mehr als einer Richtungsfahrbahn möglich wird. Dies ist sicherlich eine sehr herausfordernde Funktionserweiterung. Im Anhang B.2 ist deshalb skizziert, wie sie durchgeführt werden könnte.
In einem zweiten Entwicklungsschritt sollte eine Abbildung des Geländes erfolgen, wodurch die geometrischen Eigenschaften der Strecken Einfluss auf das Beschleunigungsverhalten der FahrerFahrzeuge erlangen können. Wenn diese Geländeabbildung realisiert ist, könnte darauf aufbauend eine makroskopische Simulation des Verkehrs implementiert werden.
Von eher untergeordneter Bedeutung ist für den Autor momentan die Erweiterung um zusätzliche Verkehrsmodelle, da mit den beiden Wiedemann-Modellen bereits zwei hinreichend realistische Modelle vorhanden sind. Um den Realitätsgrad der Simulation weiter zu erhöhen, könnten auch noch andere Verkehrsfahrzeugarten integriert werden.

Literaturverzeichnis

[Aim10] AIMSUN. Homepage. http://www.aimsun.com/site/content/section/4/72/, angesehen am 19.07.2010.

[Bolt95] BOLTZMANN, L. *Lectures on Gas Theory.* Dover Publications, 1995.

[BrWeWu00] BRILON, W.; WEINERT, A.; WU, N. *Verkehrstechnische Untersuchung der Verkehrsbeeinflussungsanlage A 44.* Schlussbericht zum Forschungsprojekt, Landschaftsverband Westfalen-Lippe, 2000.

[Erle07] ERLEMANN, K. *Objektorientierte mikroskopische Verkehrsflusssimulation.* Dissertationsschrift. Ruhr-Universität Bochum, 2007.

[Focus10] ONLINE FOCUS. *Verkehrsbeeinflussung: Harmonisch und sparsam unterwegs.* http://www.focus.de/auto/news/verkehrsbeeinflussung-harmonisch-und-sparsam-unterwegs_aid_503596.html, angesehen am 05.05.2010.

[GaHeRo61] GAZIS, D. C.; HERMANN, R.; ROTHERY, R. W. *Nonlinear follow the leader models of traffic flow.* Operations Research 9, 1961.

[Hard07] HARDING, J. *Modellierung und mikroskopisch Simulation des Autobahnverkehrs.* Dissertationsschrift. Ruhr-Universität Bochum, 2007.

[Helb97] HELBING, D. *Verkehrsdynamik - Neue physikalische Modellierungskonzepte.* Springer-Verlag Berlin Heidelberg, 1997.

[Kub10] KUBIK, R. *Szenenmanagement und effektives Rendering für Projekte der Landschaftssimulation am Beispiel des Straßenbaus.* Masterarbeit, Westsächsische Hochschule Zwickau, 2010.

[Kope97] KOPETZ, H. *Real-Time Systems. Design Principles for Distibuted Embedded Applications.* Springer Netherlands, 1997.

[MicCoz63] MICHAELS, R. M.; COZAN, L. W. *Perceptual and field factors causing lateral displacement.* Highway Research Record 25. 1963.

[Mich65] MICHAELS, R. M. *Perceptual factors in car following.* Proceedings of the 2nd International Symposium on the Theory of Traffic Flow 1963. OECD, Paris, 1965.

[Par10] Quadstone Paramics. Homepage. http://www.paramics-online.com/about-quadstone-paramics.php, angesehen am 19.07.2010.

[Pipes53] PIPES, L. A. *An operational analysis of traffic dynamics.* Journal of Applied Physics 24, 1953.

[Ptv10a] PTV AG. PTV Homepage. http://www.ptv.de/software/verkehrsplanung-verkehrstechnik/software-und-system-solutions/visum/, angesehen am 24.05.2010.

[Ptv10b] PTV AG. PTV Homepage. http://www.ptv.de/uploads/tx_ptvnews/images/Verkehrssimulation_VISSIM_Kreuzung.jpg, angesehen am 24.05.2010.

[Ptv10c] PTV AMERICA. PTV America Homepage. http://www.ptvamerica.com/company/client-references/, angesehen am 19.07.2010.

[PrigHerm71] PRIGOGINE,I.; HERMAN, R. *Kinetic Theory of Vehicular Traffic.* Elsevier, New York, 1971.

[Reus50] REUSCHEL, A*. Fahrzeugbewegungen in der Kolonne bei gleichförmig beschleunigtem oder verzögertem Leitfahrzeug.* Zeitschrift des österreichischen Ingenieur- und Architekten-Vereins 95, 1950

[SchLoh97] SCHNABEL, W.; LOHSE, D. ET AL, *Grundlagen der Straßenverkehrstechnik und der Verkehrsplanung.* Band 1 Verkehrstechnik, 2. Auflage. Verlag für Bauwesen Berlin, 1997.

[Spar78] SPARMANN, U. *Spurwechselvorgänge auf zweispurigen BAB-Richtungsfahrbahnen.* Straßenbau und Straßenverkehrstechnik, Heft 263, 1978.

[Theis97] THEIS, C. *Modellierung des Fahrverhaltens an Autobahnanschlussstellen. Dissertationsschrift.* Universität Karlsruhe, 1997.

[Todo63] TODOSIEV, E. P. *The Action-Point Model of the Driver-Vehicle System.* Technical Report 202A-3. Ohio State University, Columbus, Ohio, 1963.

[TreiHelb02] TREIBER M.; HELBING, D. *Realistische Mikrosimulation von Straßenverkehr mit einem einfachen Modell.* Contribution to the 16. Symposium "Simulationstechnik ASIM 2002" Rostock. PDF, auf beiliegender CD im Verzeichnis .\ *Masterarbeit*.

[TUBe10] TU BERLIN, Homepage, http://www.math.tu-berlin.de/coga/pics/vissim4.jpg, angesehen am 24.05.2010.

[VDI96] VDI. *Simulation von Logistik-, Materialfluß- und Produktionssystemen: Begriffsdefinitionen.* VDI Richtlinie 3633. Beuth Verlag, Berlin, 1996.

[Wied74] WIEDEMANN, R. *Simulation des Straßenverkehrsflusses.* Habilitationsschrift. Schriftenreihe des Instituts für Verkehrswesen der Universität Karlsruhe, Heft 8, 1974.

A Messgrafiken

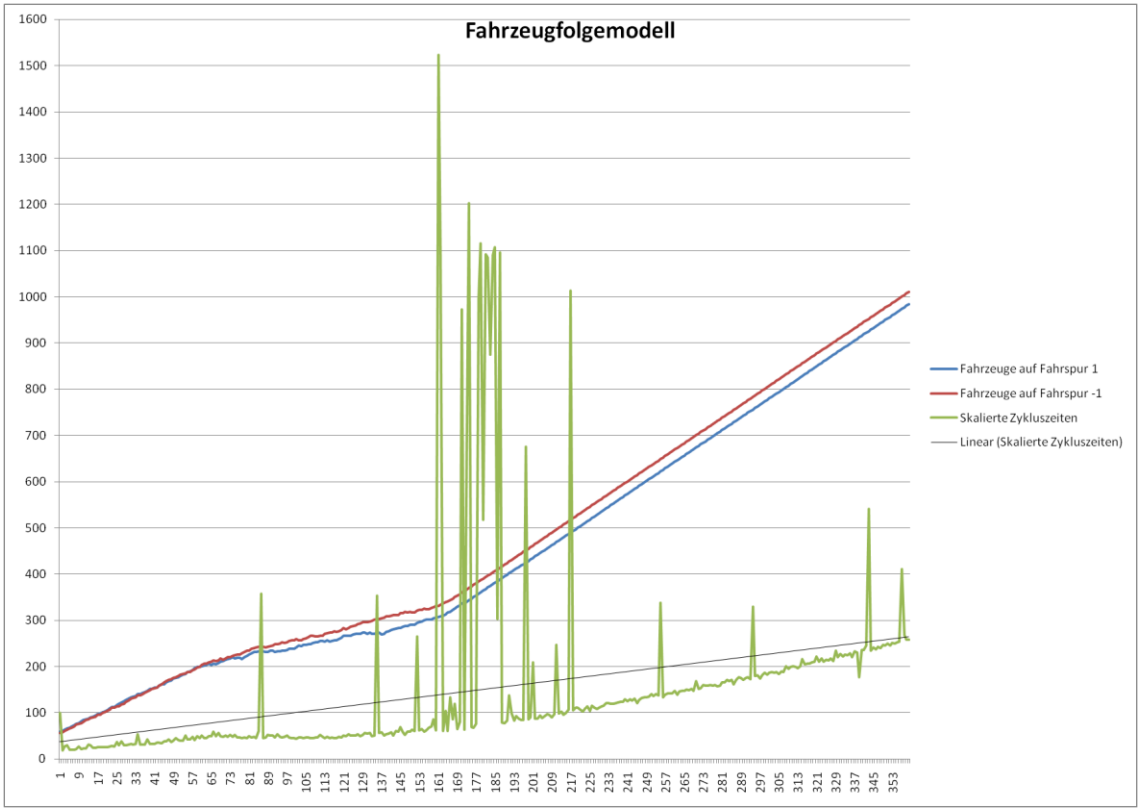

Abbildung A.1: Korrespondenz der Fahrzeuganzahl mit der Berechnungsdauer beim Fahrzeugfolgemodell

Die deutlichen Spitzen bei den Berechnungszykluszeiten in obenstehender Abbildung und in nachfolgenden Abbildungen sind durch das Scheduling des Betriebssystems zu erklären und nicht repräsentativ für die üblichen Berechnungszykluszeiten.

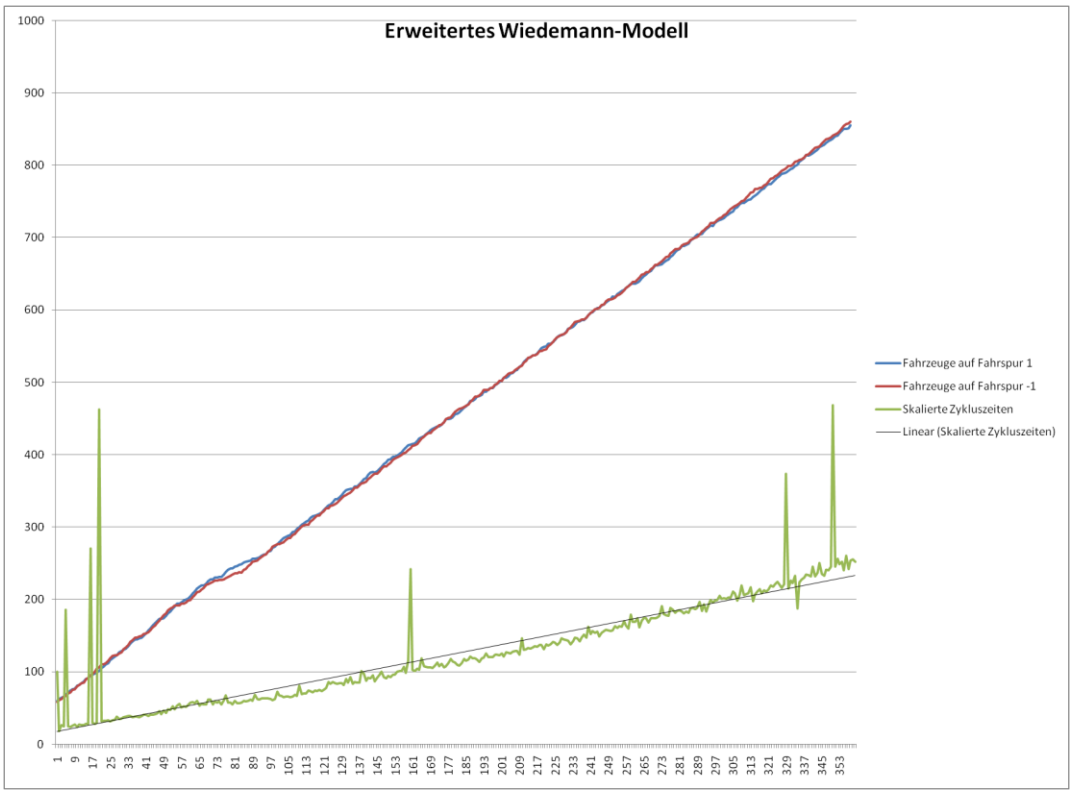

Abbildung A.2: Korrespondenz der Fahrzeuganzahl mit der Berechnungsdauer beim erweitertem Wiedemann-Modell

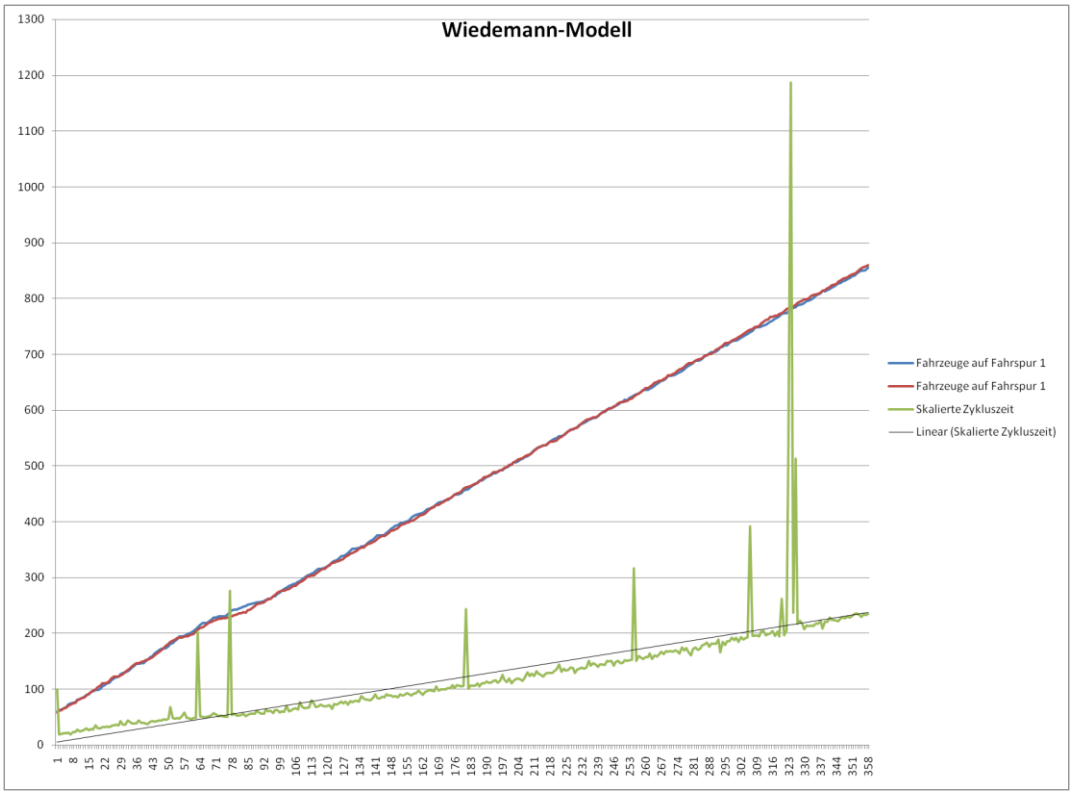

Abbildung A.3: Korrespondenz der Fahrzeuganzahl mit der Berechnungsdauer beim Wiedemann-Modell

B Mögliche Erweiterungen und Anpassungen für das Verkehrssimulationsmodul

B.1 Vorgehen für die Integration weiterer Verkehrsmodelle

Unterstützt durch ein Beispiel soll im Folgenden erläutert werden, wie ein weiteres Verkehrsmodell in das Verkehrssimulationsmodul zu integrieren wäre.
Ein Vergleich zwischen
Abbildung 3.3, Seite 46 und Abbildung 3.6, Seite 51 zeigt, dass das FahrerFahrzeug-Teilmodell und das Verkehrsmodell-Teilmodell den gleichen strukturellen Aufbau haben. Um ein weiteres Verkehrsmodell ins Verkehrssimulationsmodul zu integrieren, müsste lediglich eine Klasse, welche die konkrete Repräsentation des FahrerFahrzeuges dieses Modells beinhaltet, direkt oder indirekt von `DriverVehicle` abgeleitet werden. Beispielsweise soll die Klasse `DriverVehicleTest` direkt von `DriverVehicle` abgeleitet sein. Weiterhin müsste eine Klasse direkt oder indirekt von `TrafficCalculationModel` abgeleitet werden, welche das Regelwerk und die Berechnungsvorschrift des zu integrierenden Verkehrsmodells enthält. Die Klasse `TrafficModelTest` soll direkt von `TrafficCalculationModel` abgeleitet sein und das Beispielverkehrsmodell repräsentieren.
Damit das neue Verkehrsmodell auch verwendet werden kann, sind ferner noch einige kleine Anpassungen an anderen Stellen vorzunehmen. Für dieses neue Modell muss zunächst ein Eintrag in die Enumeration `CalculationModel` geschrieben werden. Die Bezeichnung des neuen Verkehrsmodells soll `TEST` sein (siehe Listing B.1, Seite XII). Als nächstes ist die Funktion `createDriverVehicle(…)` der Klasse `DriverVehicleSource` so zu erweitern, dass sie Instanzen des neuen `DriverVehicle`-Typs erzeugt. Zudem muss analog dazu eine Anpassung des Konstruktors von `CalculationModelThread` so erfolgen, dass er eine Instanz des neuen Verkehrsmodells anlegt. Diese beiden Schritte sind einfach durchzuführen. Es sind lediglich zwei Case-Anweisungen zu schreiben, in denen die Instanzen zu erzeugen sind (siehe Listing B.1, Seite XII). Als abschließender Schritt sind in der Funktion `updateSettingsValues(…)` der Klasse `IniFilesHandler` eine If-Anweisung zu ergänzen, in der das neue Verkehrsmodell gesetzt wird sowie ein regulärer Ausdruck entsprechend zu erweitern.

```
enum CalculationModel
{
…
   TEST
};
```

```
DriverVehicle* DriverVehicleSource::createDriverVehicle(double lanePosition,
                                                         bool carMakerCar)
{
…
   case TEST:
      return new DriverVehicleTest();
…
}
```

```
CalculationModelThread::CalculationModelThread(CalculationModel model)
{
…
   case TEST:
      this->trafficCalculationModel = new TrafficModelTest();
      this->activeModel = model;
      break;
…
}
```

```
void IniFilesHandler::updateSettingsValues(std::string filenameSettings,
                                           std::string filenameDV)
{
…
   boost::regex regexModel("SIMPLE_FOLLOW_THE_LEADER|WIEDEMANN1974|
                            WIEDEMANN1974EXTENDED|TEST");
…
   if(inputStr.compare(0, 4 , "TEST") == 0)
   {
       cm = TEST;
   }
…
}
```

Listing B.1[66]: Vorzunehmende Anpassungen für die Integration eines neuen Verkehrsmodells

B.2 Skizzierung der Integration eines Spurwechselmodells

Bevor erklärt wird, wie Spurwechselmodelle in das Verkehrssimulationsmodell integriert werden könnten, erfolgen zunächst noch Erläuterungen bezüglich des externen Fahrzeuges und des Visualisierungsmoduls. Damit die Spurwechsel des externen Fahrzeugs berücksichtigt werden können, müssen weitere Informationen vom Kommunikationsmodul übermittelt werden. Diese sind einmalig die Fahrspurbreiten und laufend die Querposition des externen Fahrzeuges zu einem Bezugspunkt, zum Beispiel rechte äußere Straßenkante. Aus diesen Informationen lässt sich dann die Zuordnung dieses Fahrzeuges zu einer Fahrspur bestimmen.

[66] Die „…“ repräsentieren den weiteren Quellcode in den entsprechenden Funktionen.

Alternativ könnten die Fahrspurbreiten auch über eine Initialisierungsdatei angegeben werden.
Auch die zu sendenden Informationen an das Visualisierungsmodul müssten so erweitert werden, dass die FahrerFahrzeuge kontinuierlich von einer Fahrspur auf die andere wechseln.
Prinzipiell sieht der Autor zwei Möglichkeiten, das Verkehrssimulationsmodul um Spurwechselverhalten zu erweitern. Bei der ersten wären die Klassen *CalculationModelThread* und *TrafficCalculationModel* um eine Funktion *doLaneChange(vector<Lane*>* lanes)* zu erweitern. In dieser Funktion würde dann in allen abgeleiteten Klassen das Spurwechselverhalten implementiert sein. In *ModelManager* würde, nachdem die Berechnungen der FahrerFahrzeuge abgeschlossen sind, die *doLaneChange(…)* Funktion von einer *CalculationModelThread* Instanz aufgerufen. Diese Struktur ist nach Ansicht des Autors allerdings nicht nur wegen der unsauberen Funktionsaufrufstruktur zu verwerfen. Es wären auch nur schwer verschiedene Spurwechselmodelle realisierbar und es käme zu einer hohen Redundanz aufgrund der mehrfachen Implementierungen, außerdem würde die Änderungsfreundlichkeit stark sinken.
Eine weitaus bessere und elegantere Variante ist die zweite Möglichkeit. Analog zu den Strukturen für die Abbildung der FahrerFahrzeuge und der Verkehrsmodelle, wird eine für die Spurwechselmodelle geschaffen (siehe Abbildung B.1).

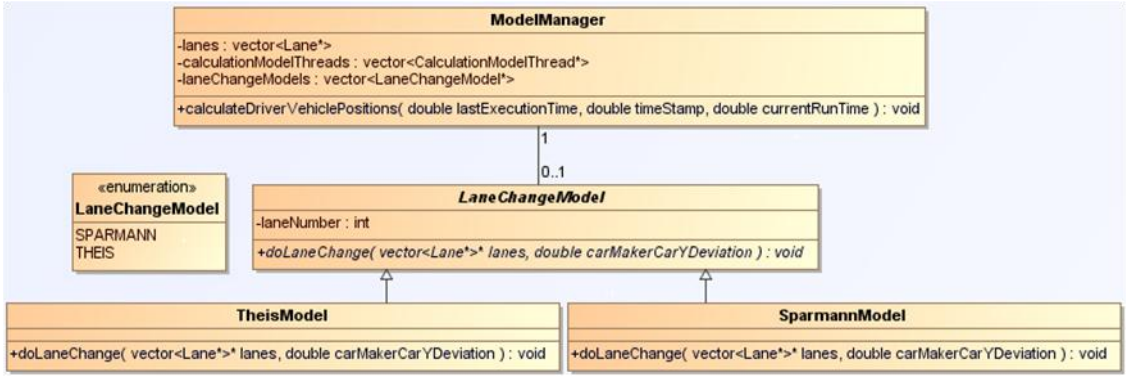

Abbildung B.1: Vereinfachte beispielhafte Modellierungsstruktur für Spurwechselmodelle

Von der abstrakten Klasse *LaneChangeModel* sind die konkreten Implementierungen von Spurwechselmodellen, wie *TheisModel* und *SparmannModel* (siehe Abbildung B.1), abgeleitet. Diese implementieren die Funktion *doLaneChange(…)* von *LaneChangeModel*, in denen dann die Spurwechselberechnungen vorgenommen werden. Die Klasse *ModelManager* würde dann in der Member-Variablen *laneChangeModels* Zeiger auf die verwendeten Spurwechselmodelle halten. Es können deswegen mehrere Spurwechselmodelle eigesetzt werden, da ein einziges möglicherweise nicht alle Verhaltensmöglichkeiten abbildet. Beispielsweise deckt das Sparmann-Modell (siehe Abschnitt 2.3.4.2) nur Überholvorgänge auf gleichen Richtungsspuren auf der freien Strecke ab und das Theis-Modell (siehe Abschnitt 2.3.4.3) nur das Spurwechselverhalten an Anschlussstellen. Bei der Berechnung der FahrerFahrzeuge würde dann in *ModelManager* nach Abschluss der Berechnungen durch die Verkehrsmodelle (zum Beispiel

Wiedemann-Modell) die *doLaneChange(…)* Funktionen der Spurwechselmodelle aufgerufen.

Die vorgestellte Spurwechselmodellstruktur aus Abbildung B.1, Seite XIII gilt allerdings nur für eine Single-threaded Abarbeitung. Für die Parallelisierung der Spurwechselberechnungen wird ein fahrspurorientierter Ansatz vorgeschlagen, dessen Struktur anlog zu der der Verkehrsmodelle ist (siehe Abbildung B.2). Dabei kümmert sich jeder Thread um die Berechnung einer Fahrspur. Die ID dieser Fahrspur wird in einer Member-Variablen der konkreten Spurwechselmodellklasse gespeichert. Da allerdings auch andere Fahrspuren bei der Berechnung betrachtet werden müssen, ergeben sich Nebenläufigkeitsprobleme. Um diese zu beheben, müssten die Zugriffsmethoden von *Lane*, *DriverVehicle* und die von *DriverVehicle* direkt oder indirekt abgeleiteten Klassen thread-safe gemacht werden. Dies ließe sich gut mit den Mutex Implementierungen von boost gestalten.

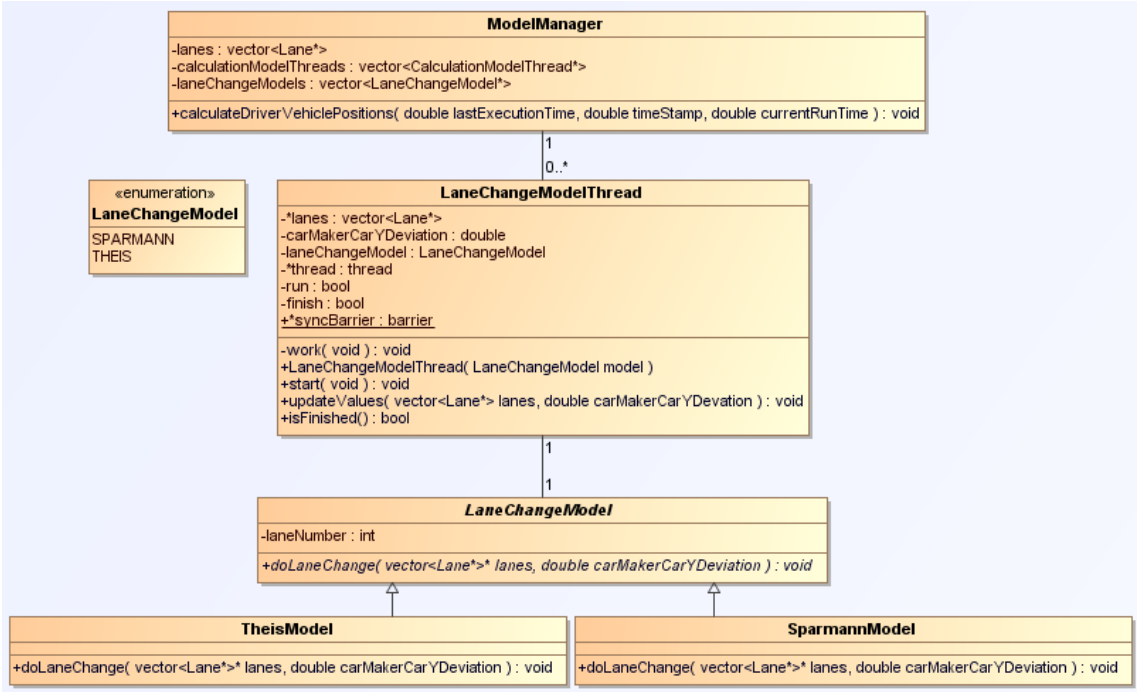

Abbildung B.2: Vereinfachte beispielhafte parallelisierte Modellierungsstruktur für Spurwechselmodelle

Durch die Spurwechselmodelle wird es notwendig werden, die FahrerFahrzeuge repräsentierenden Klassen um weitere Variablen zu erweitern. Da sich die hinzukommenden Variablen aber bei den einzelnen Spurwechselmodellen unterscheiden können, wird die Modellstruktur in Abbildung B.3, Seite XV vorgeschlagen. Von der abstrakten Klasse *LaneChangeVars* sind Klassen abgeleitet, welche die spezifischen Variablen und Zugriffsfunktionen für die Spurwechselmodelle enthalten. *DriverVehicle* wird um die Variable *laneChangeVars* erweitert, in denen die Zeiger auf die konkreten *LaneChangeVars* Instanzen wie *LaneChangeVarsSparmann* oder *LaneChangeVarsTheis* gehalten werden.

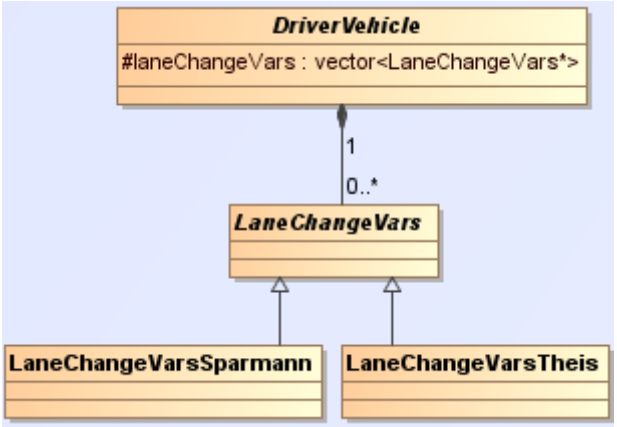

Abbildung B.3: Vereinfachte beispielhafte Modellierungsstruktur für die Erweiterung von *DriverVehicle* um Variablen für Spurwechselmodelle

C Probanden Fragebögen

Fragebogen

Angaben zum Probanden

Proband: 1

Alter: 27

Geschlecht: ☒ m ☐ w

Führerschein: ☐ nein ☒ ja

Wenn ja: ☒ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- Zuviel abstand zwischen den Fahrzeugen, nicht realitätsnahe.
- Keine Überhohlmöglichkeiten
- Verkehr reagiert nicht, um Zusammenstösse zu verhindern

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☒	☐	☐	☐

Freie Beschreibung

- Fahrzeug hinter mir hat mich durch mich überholt

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- abstand der Fahrzeuge zueinander besser
- Reaktion beim überhohlen nicht getestet

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Rückwärtiger Verkehr überholt nicht (besser als im erstem Modell)

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

-weitere abstände zwischen den Autos

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

-wie zweites Modell

Gesamtbewertung

Modell 1 unrealistisch von den Abständen zwischen den Autos und LKWs

Modell 2 und 3 keine erkennbaren unterschiede vom verhalten der KFZs

Modelldarstellung insgesamt: Autos zum teil mit transparentem Kühlergrill; unrealistische Beschleunigung (feed back fehlt)

Fragebogen

Angaben zum Probanden

Proband:	2
Alter:	25
Geschlecht:	☒ m ☐ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☒	☐	☐	☐	☐	☐

Freie Beschreibung

Zu gleichmäßige Abstände der Fahrzeuge - keine Gruppenbildung

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☒

Freie Beschreibung

Keine Überholvorgänge durchgeführt, dadurch kein Eindruck weiterer Fahrzeuge, außer dem vorausfahrenden, dieses fuhr sehr gleichmäßig

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Perfekt

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Perfekt

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Perfekt

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☒	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Perfekt

Gesamtbewertung

Durchaus realistische Darstellung des Verkehrs bei allen Modellen - außer der gleichmäßige Abstand bei Modell 1

Fragebogen

Angaben zum Probanden

Proband:	3
Alter:	22
Geschlecht:	☐ m ☒ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☐	☒	☐	☐

Freie Beschreibung

Lkw´s fuhren definitiv zu schnell.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☐	☐	☒	☐

Freie Beschreibung

Seltsame Abstände. Abstände waren viel zu groß. Geschwindigkeit ebenfalls seltsam.

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Besser als die vorherigen Modelle. Kommt der Realität nah.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☒	☐	☐	☐

Freie Beschreibung

Lkw´s sind zu schnell. Vor allem bergauf.

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Gefällt mir gut. Wie in der Realität. Bestes Modell von allen.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Autos manchmal ziemlich langsam, aber im großen und ganzem ok.

Gesamtbewertung

Fragebogen

Angaben zum Probanden

Proband:	4
Alter:	27
Geschlecht:	☒ m ☐ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☒	☐	☐	☐	☐	☐

Freie Beschreibung

- Fahrzeuge kamen auffällig gleichmäßig.
- Keine Fahrzeuggruppen → Stau erscheint unwahrscheinlich
- Geschwindigkeit wirkt sehr gleichmäßig

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- wirkt realistisch
- Überholen der anderen Verkehrsteilnehmer wäre schick

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- realistischer als im Fahrzeugfolgemodell
- ab und zu Grüppchen
- starker Gegenverkehr → überholen kaum möglich
- abbremsen von Gegenverkehr in Gefahrsituationen nicht erkennbar (riskantes Überholen)
- Ständig Verkehr → kaum Phasen ohne Gegenverkehr

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- Stau ist möglich
- Fahrzeuge achten nicht auf Hintermann (drängeln nicht möglich)
- Stellenweise unerklärliche Beschleunigung der Gruppe vor dem eigenen Fahrzeug
- Fahrzeuggruppen lösen sich bei Beschleunigung nicht auf.

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- realistischstes Modell
- ab und zu kein Gegenverkehr → sehr realistisch
- Gruppen von Autos kommen entgegen → langsamere Autos bremsen andere aus
- Keine Reaktion auf riskantes Überholen

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

- Kolonnen werden deutlicher als im vorherigen Modell, sonst keine Unterschiede erkennbar

Gesamtbewertung

Fragebogen

Angaben zum Probanden

Proband:	5
Alter:	24
Geschlecht:	☒ m ☐ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☒	☐	☐	☐

Freie Beschreibung

Die Fahrzeuge kamen in gleichen Abständen, kein Überholmanöver möglich

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Schleichender LKW sehr realitätsnah

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Anfangs zu viele LKW´s und zu wenig PKW´- Kolonnen dahinter

Bei PKW- Kolonnen fehlende LKW´s

Abstände sehr realitätsnah

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Hatte den Eindruck das alle Auto´s und LKW´s gleich schnell gefahren sind

Überholmanöver möglich

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Empfand es sehr realistisch da es auch zeitweise keinen Gegenverkehr gab

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☒	☐	☐	☐	☐	☐

Freie Beschreibung

Hatte den Eindruck das auf meiner Spur viel weniger Fahrzeuge fuhren

Gesamtbewertung

Fragebogen

Angaben zum Probanden

Proband:	6
Alter:	24
Geschlecht:	☒ m ☐ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☐	☐	☐	☐	☒	☐	☐

Freie Beschreibung

Der Gegenverkehr sah nicht besonders real aus. Es gab keine Überholmöglichkeit, weil alle Autos mit fast dem gleichen Abstandgefahren sind.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Das hinterher fahren auf meiner Spur war relativ realistisch.

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Den Gegenverkehr empfand ich als sehr realistisch, da es immer wieder Autogruppen gab und dazwischen große Lücken zum überholen.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Der Verkehr auf meiner Spur war auch realistisch. Das einzige was störte war das keiner Überholt hat.

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Kein unterschied zum anderen Wiedemann-Modell.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☒	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Kein unterschied zum anderen Wiedemann-Modell.

Gesamtbewertung

Insgesamt fand ich das 2. Und 3. Modell am realistischsten.

Fragebogen

Angaben zum Probanden

Proband:	7
Alter:	24
Geschlecht:	☒ m ☐ w
Führerschein:	☐ nein ☒ ja
Wenn ja:	☐ A ☒ B ☐ C ☐ D ☒ M

Fragen zu den Testläufen

Wie realitätsnah empfanden Sie den Gegenverkehr beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☐	☒	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Abstand der Fahrzeuge zueinander oft zu groß; LKW's ziehen zu wenig aufgefädelte Fahrzeuge hinter sich her; kaum Überholmöglichkeiten was in der Realität nicht in der Form der Fall ist (Außnahmen – stark frequentierte Ferienlandstraße an typischen Anreise- und Abreisetagen)

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Fahrzeugfolgemodell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Fahrzeug schließt nicht auf vorausfahrenden LKW auf (der in Realität oft langsamer ist als PKW); oftmals ist Beschleunigung und Tempoverringerung nicht bemerkbar

Wie realitätsnah empfanden Sie den Gegenverkehr beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Gegenverkehr insgesamt realitätsnaher, Abstände zwischen zwei großen Gegenverkehrsgruppen realistisch – daher auch realitätsnahere Überholmöglichkeiten; Langsame Verkehrsteilnehmer ziehen schon eher eine „Schlange“ hinter sich her – obwohl es erfahrungsgemäß in der Realität mehr Fahrzeuge sind.

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☐	☐	☒	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Geschwindigkeit großer Verkehrsteilnehmer (LKW) kaum von PKW unterscheidbar; kein Gefühl für Geschwindigkeit daher auch Überholvorgang schwierig; PKW schließen zu wenig auf große Verkehrsteilnehmer (LKW) auf

Wie realitätsnah empfanden Sie den Gegenverkehr beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	9	8	7	6	5	4	3	2	1	kann ich nicht beurteilen
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Kritikpunkte der beiden zuvor getesteten Modelle nur noch rudimentär vorhanden; vor allem sind unterschiedliche Geschwindigkeiten besser wahrnehmbar

Wie realitätsnah empfanden Sie den Verkehr auf der eigenen Spur beim erweiterten Wiedemann-Modell?

sehr realitätsnah *gar nicht realitätsnah*

10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**kann ich nicht beurteilen**
☐	☒	☐	☐	☐	☐	☐	☐	☐	☐	☐

Freie Beschreibung

Verkehr kommt an die Realität ran; Beschleunigung der anderen Verkehrsteilnehmer oft zu hoch ist da bei eigenen Bremsvorgängen oft Probleme bestehen wieder aufzuschließen – man denkt (nur in dieser speziellen Situation) man fährt Moped und der vorausfahrende Verkehr hat Porsche Motoren

---

Gesamtbewertung

Das dritte Modell (erweitertes Wiedemann – Modell) ist insgesamt am realistischsten. Modell zwei (Wiedemann) kommt nur in gewissen Situationen an die Realität ran. Hauptkritikpunkt ist, dass die LKW und die nachfolgende Fahrzeuge hier noch zu weit von der Realität entfernt sind (vor allem Geschwindigkeit und Beschleunigung) Modell eins ist wenig realistisch.

Zu kritisieren ist vor allem noch, dass alle anderen Verkehrsteilnehmer nicht überholen. Ferner fehlt etwas das Gefühl für die Geschwindigkeit.

D Vollständiges Klassendiagramm des Verkehrssimulationsmoduls

CD-Inhaltsverzeichnis

1) Masterarbeit
- *a) Das PDF-Dokument der Quelle [TreiHelb02].*
- *b) Das ausführliche Klassendiagramm des Verkehrssimulationsmoduls.*
- *c) Diese Masterarbeit als PDF-Dokument.*
- d) ScreenshotsWebQuellen
  - *i) Screenshots der Webquellen vom Tag der Ansicht.*

2) Tests
- a) ProbandentestsVerkehr
  - i) Fragebögen
    - *(1) Auswertungsdatei zu den Fragebögen.*
    - *(2) Fragebögen der 7 Probanden.*
  - ii) Testdaten
    - *(1) MATLAB M-File zum plotten der Trajektorien.*
    - *(2) Auswertungsdatei zu den Geschwindigkeiten der Probanden.*
    - *(3) Ordner mit den aufgezeichneten Daten der Testläufe.*
- b) SimulationstestsVerkehr
  - i) SFTL
    - *(1) Datei mit Rohdaten und Initialisierungsdateien.*
  - ii) W1974
    - *(1) Datei mit Rohdaten und Initialisierungsdateien.*
  - iii) W1974Ex
    - *(1) Datei mit Rohdaten und Initialisierungsdateien.*
    - *(2) MATLAB M-File zum plotten der Trajektorien.*
- c) Simulationszeitdauertests
  - *i) Auswertungsdatei zu den Simulationszeitdauern.*

Thesen

1. Das Wiedemann-Modell ist ein sehr detailliertes Verkehrsmodelle und bildet das Verhalten von FahrerFahrzeugen gut ab.
2. Die Integration des entwickelten Verkehrssimulationsmoduls in ein Verteiltes System und die Parallelisierung ermöglichen die flüssige Visualisierung des simulierten Verkehrsstroms und die Integration des externen Fahrzeuges.
3. Das entwickelte Verkehrssimulationsmodul ermöglicht, mit einem externen Fahrzeug im simulierten Verkehrsstrom zu fahren.
4. Die ausreichende Parametrisierbarkeit des entstandenen Verkehrssimulationsmoduls ist für eine vielseitige Nutzbarkeit wichtig. Das wird durch die Differenzierungsmöglichkeiten in den Initialisierungsdateien erreicht.
5. Das während dieser Arbeit entwickelte Verkehrssimulationsmodul ist zu einem Großteil plattformunabhängig implementiert. Diese wichtige Eigenschaft konnte hauptsächlich durch den Einsatz der boost Bibliotheken erzielt werden.
6. Die einheitlich modellierten Schnittstellen durch abstrakte Klassen zum Verkehrsmodell- und dem FahrerFahrzeug-Teilmodell sowie die allgemein gute Implementierung des Verkehrssimulationsmoduls ermöglichen die leichte Erweiterbarkeit um zusätzliche Verkehrsmodelle. Diese Eigenschaften sind für die eventuelle Weiterentwicklung des entstandenen Moduls von Bedeutung.
7. Die Entscheidung, die aktuelle Simulationszeitschrittdauer auf Basis der vorangegangenen Simulationszeitschrittdauer zu wählen, führt zu keinen signifikanten Problemen.
8. Das Wiedemann-Modell und insbesondere das erweiterte Wiedemann-Modell eigenen sich gut für die realistische Verkehrserzeugung.
9. Das Fahrzeugfolgemodell erzeugt einen eher unrealistischen Verkehrsfluss.
10. Die Implementierung von Spurwechselverhalten würde den Realitätsgrad der Simulation und den Funktionsumfang des Verkehrssimulationsmoduls steigern.
11. Das Verkehrssimulationsmodul bietet großes Potenzial und sollte weiterentwickelt werden.
12. Wie durch Probandenversuche bestätigt, ist es gelungen, ein Softwaremodul zu realisieren, das realistischen Verkehr simuliert.